Report No. 380

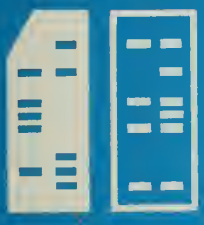
A SIMULATION STUDY OF A DISK STORAGE ALLOCATION SYSTEM

by

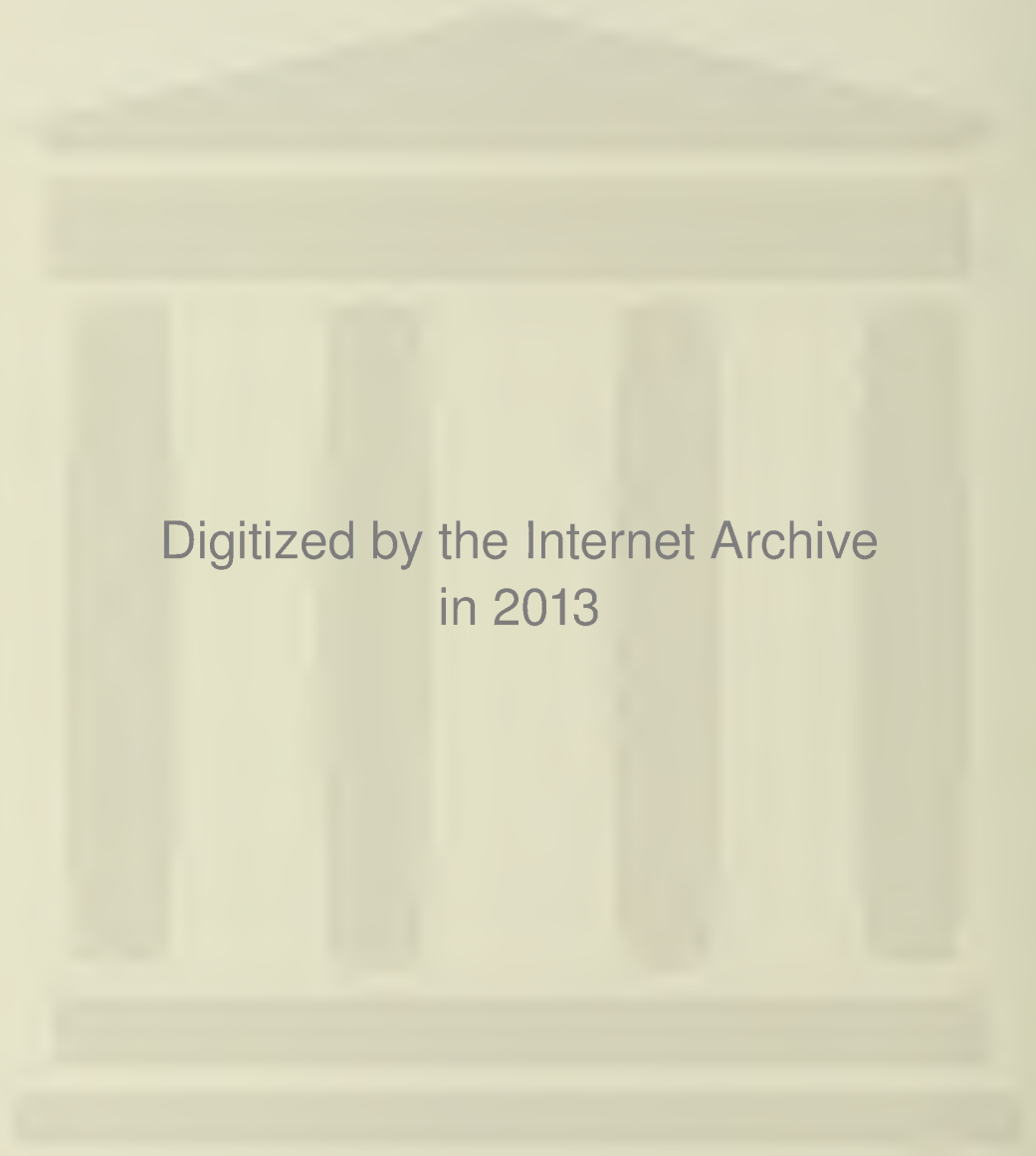
Judy Ann Lender

April 30, 1970

ILLIAC IV Document No. 210



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS



Report No. 380

A SIMULATION STUDY OF A DISK STORAGE ALLOCATION SYSTEM*

by

Judy Ann Lender

April 30, 1970

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

* This work was supported in part by the Advanced Research Projects Agency as administered by the Rome Air Development Center under Contract No. USAF 30(602)-4144 and submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science, January 1970.

ABSTRACT

This report gives a discussion of the field of simulation. The use of the simulation language SIMULA is then used to program general disk storage allocation simulators with application for use in testing allocation algorithms for the ILLIAC IV disk file allocator. Finally the concept of system design by using simulation in the design phase at various design levels is presented, with emphasis on using SIMULA for design from the hardware level upward.

ACKNOWLEDGEMENT

The author would like to express her gratitude to Dr. R.S. Northcote and other members of the ILLIAC IV staff, for their valuable assistance and encouragement in this work.

The encouragement and advice from the ILLIAC IV Operating System staff and Mr. Marvin Graham is especially deeply appreciated.

Thanks also goes to Mrs. Kay Flessner for the typing of this manuscript.

Finally, the author would like to express her appreciation to the Department of Computer Science for making this work possible.

TABLE OF CONTENTS

1. INTRODUCTION

There are basically three purposes for simulation: (1) studying the behavior of a system, (2) training purposes, and (3) systems design. By far, the most extensive use of simulation has been in the first two areas.

A simulator built for either or both of the first two purposes has always been built subsequent to or concurrent with the system which it is simulating. Therefore, the simulator is well-defined in the sense that the simulated system has undergone fairly extensive definition. In systems design, however, this order is reversed and a simulator is built before the design is finalized. This is a simulator of system behavior rather than a simulator of well-defined physical components, and the simulator itself is used as a tool to bring the system to a well-defined state. Of course, studying the behavior of a working system often leads to modifications in the design of that system also.

Much of this paper will describe a simulator designed for the first purpose mentioned above; that of studying the behavior of the disk file allocator algorithm for the ILLIAC IV disk. The simulator sets up a dynamic input-output environment for the disk file allocator by simulating file requests whose entries to the system are time dependent.

The objective of this paper, however, is not so much to describe and demonstrate a few lines of code, but to disclose the disciplines involved in model building and systems design. For this reason, the simulator mentioned above will be discussed only for the purpose of demonstrating the discipline of model building and simulator design. The paper will be concluded with a discussion of the use of simulation in systems design, including a short review of the literature and, finally, this author's own proposal for the use of SIMULA for this purpose.

2. MODEL BUILDING AND SIMULATION LANGUAGES

2.1 Philosophy of Model Building

Basic to every concept discussed in this paper is the notion of a "system", for it is a "system" that will be modelled and simulated. Therefore, it is necessary to define precisely what is meant by the term and related terms within the scope of this paper.

A system is a connected set of components whose behavior is self-determined. By self-determined it is meant that the state and outputs of the system can be predicted from knowledge of the previous state of the system and the inputs. A component is a device or program which is self-determined and may, in fact, be a system itself. By connected it is meant that the inputs to some of the components may be the outputs of other components of the system. A system may be a component of another system, so the distinction between system and component is one of level depending upon the system definition of the researcher doing the modelling. The most important and necessary requirement of all components defined for a particular system is that they have well-defined behavior, or it must be possible to find an algorithm which describes their behavior.

The above definitions conform to those used in the literature and, while there may be systems which do not fit the above definition, computer systems, either hardware or software, do fit the definition. It is clear that these definitions deal with the model, or static representation, of the system. The following are definitions that refer to the dynamic aspects of a simulation, or model manipulation, in order to achieve a "motion picture" of reality. The definitions will be extended later in connection with the discussions on simulation languages.

An event is an active instance of a component in time. For example, if the component happens to be a device, the event is the manipulation of that particular device as it occurs in a dynamic relationship with the other components in the system. If the component in question should be a program, the event is the execution of that program as it occurs in relation to the rest of the system. The concept of an event is one basic to any discussion of simulation and simulation languages.

This author wishes to define the term simulation in the following way, which hopefully will aid the reader in visualizing a static representation of a dynamic situation. First, however, two preliminary definitions are in order. A linear time relationship among events is one in which the events occur in increasing time intervals (simultaneity is not implied) or nondecreasing time intervals (simultaneity is implied). An event set is an ordered collection of events which exist in an increasing time relationship to one another. Finally, a simulation is a set of event sets.

Underlying the construction of any model of a real world system is a certain "way of looking at things" or discipline necessary to facilitate the description of the model. It will be seen later that simulation languages are constructed around this discipline. The remainder of this chapter is a presentation of the philosophy of model building, as applied to computers.

It should be fairly obvious by the preceding discussion that the first problem is to define the separate components of the system under investigation. These components are either fairly evident by their physical nature or are determined on the basis of their behavior relevant to the particular aspect of the system being studied.

In illustration of the criteria for determining individual components of a system, consider a dynamic storage allocation system. A general problem of the allocation type can be described by four characteristics: (1) receipt of the request, (2) allocation of the request, (3) use of the allocated space, and (4) release of the space. No dynamic storage allocation problem could be described with the omission of any one of these characteristics and, therefore, they constitute the components of the model.

Construction of the model will begin with defining the precise behavior of each of these components with respect to the particular allocation system under consideration. For example, if the times of arrival of requests are random, as is generally the case, then a queue must be added to the model to hold requests until allocation can take place.

The next problem encountered after the components of the system have been defined are the interactions of these components with one another. All relevant component interactions must be defined. For example, the component which receives the request in the dynamic storage allocation problem interacts directly with the component which performs the allocation by initiating or "calling" it, and also passes data (the request) to that component.

Two individual researchers might view a particular system differently, and it should be apparent that the definitions of the components of a particular system are by no means fixed and absolute. They exist "in the mind of the beholder" so to speak. A different way of viewing the dynamic storage allocation problem might be the following. Since allocation of this type is essentially the manipulation of a free space map, or list, the components might be defined as follows on a more physical (or structural) rather than a functional (or behavioral) basis: (1) the queue, (2) the free space list, and (3) the in-use list. Descriptions of the interactions of these components with one

another would consist of algorithms changing the physical contents of one component based on the input of another, rather than a change of state, as with a more functionally-defined component.

In summary, to construct a model of any physical system, it is necessary to determine the separate components and to describe the interactions of each of these components with the other components in the system. More specifically, these components perform actions and they are data carriers for certain other components of the system.

Therefore, definition of each of the components of the system and a description of the actions and data handled by each component will constitute a description of the model to be constructed and the simulator itself. This is the rationale behind most simulation languages; that a component is described by a discrete entity usually known as a process and that a system description in terms of the specific simulation language involved also constitutes the source code of the simulation program.

2.2 Design of Simulation Languages

2.2.1 General Consideration

One definition of a simulation model is the following: the representation of a dynamic system in a form suitable for manipulation by a computer. The key word here is "dynamic" meaning a change of state as a function of time. In defining the components of the model to be simulated, it was not necessary to consider this aspect of a simulation model. However, this aspect constitutes the major problem in the design of all simulation languages.

To get an idea of the factors determining the design of a simulation language, consider a set of events in the real world. First there is an interdependence among the events. Every real world event depends on the events preceding it and, in turn, will affect those events which succeed it.

At this time the author would like to extend the definition of an event set, keeping in mind that an event is one specific instance of a process. An event set is an ordered set of events, all members not necessarily distinct from those of a separate event set, but with the property that if one event is a member of two event sets, then all succeeding events are also members of both sets. Each event of an event set is directly activated by the preceding event and, in turn, directly activates the succeeding event. This is implied by the fact that, as stated previously, the ordering is linear by increasing times.

Second, the time-ordered occurrence and interdependency of members of an event set illustrate continuity in the set of real world events.

Third, each event is unique in that it occurs once and only once in the real world.

Finally, the real world is distinctive for its simultaneity or, in terms of the definitions given so far in this paper, it gives the appearance of being a set, or system, of event sets utilizing a common time stream.

Any simulation language must be able to deal effectively with these four aspects: interdependence, continuity, uniqueness, and simultaneity. The next section will describe how SIMULA, an ALGOL-based simulation language, handles these aspects.

2.2.2 Discussion of SIMULA

Consider first the two basic problems which must be handled by any simulation language, namely, component description and the handling of the dynamic aspects of a simulation.

SIMULA describes a component of a system as a class of processes which are described programmatically by an activity declaration, but which allows for the inclusion of SIMULA sequencing statements (to be defined later). A process can be in one of four possible states; active, suspended, passive, and terminated. If a process is active, it is currently being executed in the simulation. These states will be discussed in more detail later.

Processes can be referenced individually by a pointer to an area of memory containing the data local to the process and some additional information defining its current state of execution. These pointers are called elements. In fact, if a process must be referred to after its initial activation, the reference must be through an element representing that process. Formally, an element is declared to be of type element.

Example:

```
element pat;
activity secretary (redhaired, thumbs);
         boolean redhaired; integer thumbs;
begin. . .end;
pat:=new secretary (true,10);
```

There are two distinctions to be noted here. A process is one dynamic instance of an activity declaration which includes any of the four possible states. An event is one active instance of a process and is considered to take place in one instant of simulation system time (SST) since the computer can perform only one state change at a time, and updating the SST would itself be such a change.

The dynamic aspect of the simulation is controlled by the sequencing set (SQS) which is a set of event notices for which events have been scheduled but not completed. An event notice contains a reference to the associated process (through an element) and a real number, called its time reference. The SQS is ordered according to nondecreasing time references. The event notices at the "lower end" of the SQS is called the current event notice; its associated process is the currently active one and its time reference is regarded as the current value of the simulation system time.

Event notices in the SQS are manipulated by statements called sequencing statements. One cannot write a simulator in SIMULA or any other simulation language for that matter, nor can one satisfactorily describe a system to be simulated unless one has a clear understanding of the sequencing statements and how they affect the SQS. Similar constructs are used in other simulation languages.

Before describing the sequencing statements, it is necessary to describe in further detail the four possible states:

(1) active--the currently active process can, by sequencing statements, alter the states of processes, including its own.

(2) suspended--a suspended process has an associated event notice and a reactivation point, the point at which control reenters the process at the time of its next active phase, namely the place where it most recently left its previously active phase. Unless a change of state is caused by another process, the next active phase of this process will start when the event notice becomes the current one.

(3) passive--a passive process has a reactivation point, but no associated event notice. It will remain passive until a change of state is caused by another process. At the time of generation a process is passive, and its reactivation point is in front of the first statement of its operation rule.

(4) terminated--a process becomes terminated whenever control passes through the final end of its operation rule. Such a process has no reactivation point and no event notice. The state of a terminated process cannot be altered by ordinary sequencing statements. A process will remain part of the system at least as long as it has an associated event notice.

Sequencing statements are statements operating on the SQS, thereby altering the states of processes. A sequencing statement may delete an event notice and/or schedule an event by generating an event notice. The basic sequencing statements are:

(1) cancel (<element expression>), which deletes the event notice, if any, associated with the referenced process;

(2) terminate (<element expression>), which in addition to deleting the event notice, also deletes the reactivation point, if any;

(3) scheduling statements, which usually generate an event notice for a specified process and include it in the SQS; in addition, the statement will specify explicitly either the time reference of the event notice, or its position in the SQS.

The basic scheduling statements are special syntactic constructs as follows:

activator ::= activate | reactivate

simple timing clause ::= at <arithmetic expression> |
delay <arithmetic expression>

timing clause ::= <simple timing clause> | <simple timing clause> prior

<scheduling clause> ::= <empty> | <timing clause> | before <element expression> | after <element expression>

<scheduling statement> ::= <activator> <element expression> <scheduling clause>

The activator activate will cause the generation of an event notice only if the referenced process is passive, whereas reactivate in addition will delete the event notice associated with an active or suspended process and "reschedule" the event. A timing clause specifies the time reference of the generated event notice and this determines its position in the SQS. The event notice is normally placed behind all others with the same time reference, but it can also be placed in front of these event notices by adding the symbol prior.

The reader is referred to the paper by Dahl and Nygaard [1] and the manual on the B5500 implementation of SIMULA [2] for a more complete description of the language.

Now, consider the four characteristics of a real world system. The SIMULA construct which handles each one will now be discussed.

First, there is the characteristic of interdependence. It was mentioned earlier that a process has two aspects; it is a data carrier and it performs actions. SIMULA has a construct called a connective statement which allows a process to access the local variables of another process. Therefore, not only can one process "schedule" another process or otherwise change its event notice in the SQS, but a process can also pass parameters to another process as one would with Algol procedures, or access the local variables of another process through the connective statement.

The SQS, if all of its contents could somehow be saved throughout the simulation, would clearly be a set of event sets as defined in the preceding section. This implies that it is ordered by nondecreasing simulation system times. Continuity is maintained in that processes are activated and become current in the order of nondecreasing SSTs in the SQS.

The uniqueness of an event is inherent in its SIMULA definition, an active instance of a process which occurs once and only once throughout the simulation.

Two or more event notices can exist in the SQS with the same time reference and, even though the processes referenced by these event notices are executed by their positions in the SQS, as far as the simulation is concerned, they occur simultaneously.

3. USE OF SIMULATION FOR STUDYING CERTAIN ASPECTS OF A SYSTEM

3.1 Discussion

By far the most extensive use of simulation is in the area of studying the behavior of certain aspects of a fairly well-defined system. This implies that many hours were spent in design in conferences and at the drawing boards, and also in the fabrication of the actual system before consideration was given to a simulation study.

In constructing a simulator for this purpose, the component being studied should exist as a procedure, identical in every way to the real world component of the system; those components which directly interface with the component or constitute the input to the component under study may be required also to be procedures, or at least contain parts of the actual code of the real system. This constitutes the most important and most difficult part of simulator design, that of determining realistic driving for the component and relevant input based on the researcher's own specific purpose for the study.

If unnecessary or infeasible to give a real world definition to the components interfacing with the component under study, it is usual to design an activity giving a statistical representation of the behavior of that component. For this reason, most simulation languages include several procedures which generate random numbers according to various distributions.

3.2 The ILLIAC IV Disk File Allocator Simulator

3.2.1 The Component to be Studied--The Disk File Allocator Procedure

According to Mills and Alsberg, [4], allocation of the ILLIAC IV disk among jobs requesting space is a non-trivial programming task due to phase and channel relationships which can be specified by the programmer.

ILLIAC IV programs will be heavily I/O bound and disk latency must be minimized. Therefore, it is essential that some means be devised by which allocation algorithms can be easily tested and revised. Such a means is obviously a simulator. In the following paragraphs a simulator will be constructed according to the criteria established in earlier sections.

The Disk File Allocator takes a specific request for disk space and attempts to allocate this space by utilizing a map of free disk space. It is a modular component of the ILLIAC IV Operating System. Thus, the allocator itself exists in the simulator as a procedure, PROCEDURE ALLOCATE, in the actual coded form that it has in the operating system. It should be obvious at this point that if the real world system has a modular design, definition of some of the phases of simulating that system will be quite trivial. Another system, whose component boundaries are not well-defined might prove to be a little more difficult, and would require extra care and definition on the part of the researcher in order that no relevant factors be left out.

The allocator takes its request data from two arrays that are designed on the concept of levels of blocks. This structure of the request data is called the allocation tree. Allocation begins with the highest level in the disk hardware hierarchy, the electronic units (EUs), allocates them, and then descends through the rest of the hardware levels, the storage units (SUs), tracks (TRKs), and segments (SEGs). The blocks are arranged corresponding to this same hierarchy with an EU level block containing the EU number, the physical or virtual bit, pointers to the SU level block, the TRK level block, and the assignment block which contains specific information as to how the space is to be allocated within segments, and an indication as to whether the request is phased or contiguous or, if neither, that it is in the "junk" category, meaning that it can be allocated anywhere there are segments available.

The allocation tree, which constitutes the direct input to the allocator itself, is created by a procedure called TREEBUILD. This procedure, at the time of the design of the simulator, had been coded and debugged. A quick analysis showed the desirability of including the TREEBUILD procedure in the simulator rather than to hard code data for the allocator in the form in which it is utilized. An allocation tree is not built for a particular request until the time for an allocation attempt.

The input to TREEBUILD is in the form of a structure called a disk file block. This block is created by the job parser from ICL file specification statements at the time the request enters the operating system. As soon as the job parser is debugged, it should be added to the simulator. This will allow complete flexibility of the input in that the disk file requests may be input as ICL statements. The simulator has been designed in order to facilitate additions such as the above. Instructions for the addition are given in a later section of this report.

3.2.2 The Structure of the Simulator

3.2.2.1 Basic Structure of a Dynamic Storage Allocator System

The "core" of the simulator has been determined; it is made up of the procedures ALLOCATE and TREEBUILD, whose actual code was lifted straight from the operating system. Next, there is the problem of defining the structure of the simulator, which is primarily devoted to driving the input to procedure TREEBUILD. The output of the simulator is left entirely to the discretion of the designer and is not a factor in a discussion of the simulator structure.

Consider a general dynamic storage allocation problem, including a storage medium with a finite number of units available for allocation and a queue of requests, each for some integer less than or equal to the total number of units available for allocation, representing the number of contiguous

units of storage requested. Following allocation, these units will be held for a finite length of time and then again be made available for allocation.

As indicated in an earlier chapter, there are four main characteristics of any allocation problem: (1) receipt of the request, (2) allocation of the request, (3) the holding of the allocated space, and (4) release of the space. Since there is no constraint upon the times of arrival of requests to the system under consideration and since allocation handles only one request at a time and requires a finite amount of time for each one, a fifth characteristic, queueing of the requests, is added to this system. In the following paragraphs, each of these characteristics will be described as functional activities in the simulator, and the data they carry and the actions they perform will be discussed.

Receipt of the request. In the real world, requests enter the system at random times. Therefore, the times of entrance are chosen randomly from a uniform distribution provided by a SIMULA procedure. Actually this characteristic is best handled in the simulator by two activities, one to generate the request and the other to receive and queue it. The main difference in concept between these two procedures is that one, the activity to receive and queue the request, is part of the system and the other, the request generator, is not. This request generator is the driver of the whole simulation. If there is a system which exists entirely in the form of real-world procedures, and if its input is in the form of a driver activity, then it is still a simulator. This fact and considerations leading to it will form the basis of a subsequent chapter.

Allocation of the Request. Functionally an allocation attempt involves the following steps. First, the queue is accessed and the information on the head element of the queue is obtained. The allocator procedure then is called and an allocation attempt occurs. If the allocation was successful,

the reference to the request is removed from the queue. The request just allocated goes into the state of utilization of the allocated space, i.e., the space is unavailable for allocation to another request and it does not exist in the free space list. Obviously, if the queue is empty, then the allocation attempt is bypassed.

The above describes two separate events, the accessing of the queue and the allocation attempt. These events could have been described as two separate activities; however, they occur during the same system time in this example. If it had been desired to increment the system time by any amount between the execution of any two separate entities within the same activity, all that is needed in SIMULA is to insert reactivate current delay <increment> prior. As mentioned previously, the symbol prior places the time reference of this generated event notice in front of all others with the same time reference in the SQS.

Holding of the Allocated Space. The behavior of this phase merely constitutes a hold through several system times and simulates the behavior of some occurrence that utilizes the allocated space. The amount of time that the space is held is, of course, generated randomly, either from a uniformly distributed stream, or from some other distribution based upon previous analysis.

Release of the Allocated Space. Since simulation of the utilization of the space is only a hold, this aspect and the calling of the procedure to release the space are combined into one activity. The hold is randomly generated as mentioned above.

As a preliminary demonstration of the use of SIMULA to construct a dynamic storage allocation simulator, an algorithm for reserving variable-sized blocks of memory from a larger storage area was used. This algorithm,

and its companion algorithm for releasing this storage area, is very similar to the first-fit and liberation algorithms of Knuth [3], except that the algorithms were modified to manipulate a separate directory of available space instead of using the available space itself to contain such a list, as done in Knuth's example. This was done because the disk file allocator of the ILLIAC IV Operating System will use a free space map and because, in dealing with allocation of disk space, the nonuniform access time makes it better to maintain a separate directory of available space. The SIMULA source code and a sample of the output are given in Appendix C.

3.2.2.2 Evolution to the ILLIAC IV Disk File Allocator Simulator

No activities were added or deleted in the transition from the small simulation of Knuth's algorithms to a simulator of the ILLIAC IV allocation process. The basic ingredients of any dynamic storage allocation scheme are no different in this case. What has changed is the behavior of some of the individual activities. These will now be discussed.

The "driver" or request generator. As work progresses in this phase of the operating system, this activity often will be changed to accept input in various ways. At the present time, the activity calls a procedure which reads cards in the format of the disk file block (created by the job parser in the actual operating system) and puts the data from these cards into a file called PASSFILE which is accessed by procedure TREEBUILD. This activity contains a loop which increments a request count and then activates the process which queues the request. The request is identified in the system by its number.

One might argue that it would be more proper for the number assignment to take place within the activity which queues the request in that it is within the system and number assignment will, of course, take place within the system. By definition the driver activity does not do any more than drive the

system. It should perform none of the activities of the system. However, in a simulation designed for study of a certain aspect of a system, the aspect itself becomes the system (from the simulator's standpoint) and its boundaries are determined completely by the most convenient interpretation for the designer.

Later versions of this activity, as mentioned in an earlier section, might include the portion of the job parser which builds the disk file block from the ICL file request statements. This would provide the most flexible input. However, short of having the job parser, a suggestion for a method of input would be to set up ten or twenty data sets each representing a file block entry for one job's file requests, put these data sets into a file, and then draw randomly from this file as the simulation proceeds. Because of the simplicity of this activity, the user of the simulator may be as creative as he wishes as far as input to the system is concerned.

Activity Queuerequest. The ILLIAC IV Operating System will have several queues, called categories, each based upon a value representing maximum time requirements for execution. This value is called a timeshard and the basic timeshard unit is a quadrant minute [4].

The simulator was designed so that the researcher may read in on cards the number of queues and the value of the timeshard of each. Activity Queuerequest puts the request in the proper queue based on the execution time estimate entered with the job.

The researcher may wish to add a priority scheme within queues as such a scheme becomes defined for ILLIAC IV.

Activity Supervisor. This activity has been modified from the simple basic example more than any other, due to the ILLIAC IV Operating System's procedure for choosing the next job to be allocated. This procedure

is as follows: The supervisor goes first to the category of queue with the smallest timeshard and checks the timeslice value of the queue. If this value is greater than zero and the queue is not empty, the supervisor tries to allocate the next job in the queue, etc. If the allocation attempts for all jobs in a particular queue are unsuccessful, or if the queue is empty, or if the timeslice for the queue is less than or equal to zero, then the supervisor moves to the next queue in the order of the increasing timeshard value. If all the timeslice values of the nonempty queues are less than or equal to zero, then the timeshard value for that queue is added to the current timeslice value. If the resulting value is greater than zero, then the timeslice value is set equal to the timeshard value. This reinitializes all queues to positive timeslices.

When a successful allocation occurs, the job is removed from the queue.

Activity Jobrunning. This activity, which simulates execution time or the time that allocated space is unavailable for reallocation, was changed only to the extent that a peaked random number distribution, peaked about a certain value, was used instead of the uniform distribution. The derivation of this procedure for drawing from a peaked number distribution is given in Appendix B.1.

It should be obvious that any level of sophistication may be reached within an activity. This, again, is an illustration of the top-down approach in the study of systems. The SIMULA source code and examples of execution output are given in Appendix D.

3.2.2.3 Discussion of the Problems Encountered in Coding the Simulator

The current SIMULA translator cannot handle ALGOL constructs such as parameterized defines and case statements, both of which are used extensively in the disk file allocator and its array defines. Therefore, the source code for the simulator exists in two parts: (1) the SIMULA untranslatable portion, consisting of the disk file allocator, its procedures TREEBUILD and ALLOCATE, and its array declarations and defines, and (2) the SIMULA translatable portion, consisting of the main SIMULA block and a few variables global to this block. This portion includes dummy declarations for those items in the untranslatable code which need to be declared to the SIMULA translator and a control section for a program called SIMULA/MERGE, which is used to merge the translated portion with the untranslatable portion prior to the ALGOL compile of the complete simulator. The translation and compilation phases of the simulator are discussed in Appendix A, and the code for SIMULA/MERGE is contained in Appendix B.2.

4. USE OF SIMULATION IN SYSTEMS DESIGN

4.1 Review of the Literature

4.1.1 The "Top Down" Approach to Systems Design

In most problem solving situations, the emphasis is first to define the problem precisely before going about devising a method of solution. However, when one considers the design of complex systems, it is not always possible to define precisely the goal of the design process. It is possible, however, to state rather definitely what functions the system is to perform, i.e., the system definition at the design stage is oriented more toward the behavioral, rather than a physical or structural, definition.

As noted in the discussion on simulation languages one factor common to all is the existence of an entity to describe a component of the system, either functional or structural. Obviously, if one has defined the behavior of the system to be designed, a simulator may be written to simulate that behavior. Then the simulated system may be broken up into components and their interconnections, specifying the behavior of each component. Then, in turn, each of these components is broken up into subcomponents with interconnections and behavior specified, and so on, until the final design level, in which units small enough to be completely designed, are obtained.

The approach just described is known as the "top down" approach to systems design and, as demonstrated, involves starting at the "top" with a complete specification of the behavior of the system and then breaking the system into smaller and smaller components until the system is specified in terms of the basic building units.

Another advantage of designing a system by the top down discipline is that it promotes modularity of the resulting system. Modular systems are

by far the easiest to debug, easiest to modify, and are by far the most aesthetically pleasing to the experienced and inexperienced system designer alike.

4.1.2 Levels of Design and Properties of Simulation Languages for Each Level

Simulation languages used for the purpose of systems design must, according to a paper by David L. Parnas and John A. Darringer [5], have certain specifications if they are to be useful. In a later article [6], Parnas indicates that the features of the language will be different, depending upon the level of design desired, level in this instance referring to levels of functional decomposition (hardware and software being indicative of a level according to this definition) rather than referring to "levels of abstraction", or the number of times a component is decomposed into subcomponents during the design phase. The word level will have both meanings in this paper and the specific meaning in context should be obvious. If the reader is confused, he is referred to the article by Parnas and Darringer [6] in which at least two paragraphs are devoted to the two meanings of the word.

Parnas and Darringer attempted to construct a language that could be used for design by simulation. In their paper [5], they give several characteristics that such a simulation language must have. However, in the later article, Parnas points out the flaws in his original language SODAS which restricted it to a very limited class of systems, that of single level computer hardware systems. In this second paper Parnas proposes some improvements and modifications to SODAS in formulating a new language, SOCS, which will allow the design of hardware and software systems of two or more levels. He refers to this computer system class as the Operating Computer Systems, since it usually includes both hardware and the software known as the operating system for the hardware. SOCS allows the design of systems in which the decision of which components are to be hardware and which are to be software may be delayed

until very late in the design. The unit Parnas selects which allows him to postpone any decisions about hardware and software is the sequential process, which he describes as a fully ordered set of events in an operating computing system that may be performed either by hardware or by software.

Parnas' languages, SODAS (Structure Oriented Description and Simulation) and SOCS (Simulations of Operating Computer Systems) and the design levels at which they are optimally used will be discussed, followed by a look at SIMULA as a potential systems design language.

The properties of the specification-design language and translator around which SODAS was designed are as follow:

(1) Designation of inputs and outputs. If these are not distinguished, then the system is likely to be overspecified (since the designer will have to produce a component that will duplicate the behavior of the algorithm on all its variables, not simply those which will be used as inputs and outputs).

(2) Combination of independently written descriptions. It must be possible to take separately written algorithms, indicate the way that inputs of one are connected with the outputs of others, without excessive worry over conflicts in names of variables, etc.

(3) Correct handling of simultaneous events. If two of the separately described components happen to be active at the same time and interact closely, the translator must correctly simulate these simultaneous events, although it is restricted to serial execution of the individual algorithms.

(4) Components which are themselves descriptions of systems. The language structure must be recursive, i.e., any system described

in the language must be acceptable as a subsystem of a system to be described in the language.

(5) Descriptions with mixed levels of detail. The design of one component may advance faster than the design of the rest of the system. It should be possible to combine a detailed description of one component with less detailed specifications of others.

(6) Mixed structural (physical) and behavioral descriptions. A structural description of a system describes it as a set of components and their interconnections; a behavioral description is an algorithm which duplicates the behavior of the system.

(7) Broad class of systems. The language must allow the description of both synchronous and asynchronous discrete systems as well as analog or continuous systems and hybrid systems.

(8) Variety of languages for component description. It is a desirable feature of a simulation system that it permit the description and simulation of systems whose components are described in quite different languages.

The main characteristics of SODAS are as follow:

A SODAS system is a set of sub-systems, each with specified inputs and outputs, together with a "wiring diagram" description of the way that the components communicate. There may be only one or any number of subsystems and the subsystems may be described in any language which has been implemented in the system, including the SODAS language itself. The simulation algorithm that is the basis of SODAS depends only on the existence of algorithms for simulating the subsystems and not at all on the language in which the algorithms were originally described. These characteristics, as far as this writer is concerned, are the "extras" that SODAS possesses as far as a

simulation language is concerned. It has, of course, all of the characteristics of the usual simulation language.

The main difficulties encountered in any attempt to use SODAS in the design of a system of two or more hardware-software levels are now discussed along with the characteristics of the language SOCS which can handle the design of such systems.

SODAS requires explicit interconnectors between components; however, it may be the case that some processes may have an explicit intercommunication because of resource sharing through global variables, and SODAS does not allow communication through global variables. There is no facility that would allow the simulation system to determine the time of an interruption of an event due to conditions not considered at earlier stages of the design without substantial modifications to a description which was actually perfectly valid for the level at which it was written. Finally, SODAS requires that all significant interconnectors between two components be specified at the time that the functions of those components are specified and that no new interconnections show up as the design progresses. In the design of operating systems, however, it is quite normal that at certain stages in a design the sequential processes may be described as entirely independent of each other, except for certain explicit attempts at communication.

Parnas lists these features that were found in SODAS but were missing in such process-oriented languages as SIMULA.

1. Ability to have a process that consists of a set of processes, e.g., recursive structure.

2. Ability to handle difficult cases of simultaneous events.

3. Ability to handle structural descriptions of hardware.

4. The "wait until" or monitoring feature proposed for SODAS.

In conclusion, Parnas describes his new language, SOCS, as being somewhat reminiscent of SIMULA with extra constructs to provide the features just mentioned. The sub-languages and connection concept would be carried over from SODAS, with a somewhat subdued role as it would be possible, but not necessary, to leave the SOCS language to describe a process. It would be desirable of the language that a SIMULA program could be run without substantial changes, though extensive surface changes might be needed.

4.2 Use of SIMULA in Systems Design

4.2.1 Discussion

This author wishes to present a method for the design of a software operating system for an existing hardware system and to show that this can be done using SIMULA as the simulation language.

The design begins, true to previous discussions, using the top-down approach. In this case, however, the behavior of the system is specified concerning the manipulation of several structural components which are the existing fixed hardware of the system. It is proposed that these physical components exist in the simulator system as processes, each simulating the behavior of its physical counterpart in the actual computer system. Unlike the other components of the simulated system these components initially are at their final stage of decomposition and are not to be modified during the design of the system.

One reason that SIMULA could not be used for design at the hardware level is that there is no specific way to declare input and output variables of activities and, therefore, there is no ability to handle structural descriptions of hardware. However, in software considerations, all interface between activities will be handled in the same way as with the procedures through the

passing of parameters and global variables and the calling of one procedure by another.

One restriction to the use of SIMULA in systems design is that the final code of the operating system be in ALGOL, the language of the simulation language. This was not a restriction as far as SODAS was concerned, providing the necessary translators were available.

The big reason why SIMULA cannot be used for design at the hardware level is that its language structure is not recursive, i.e., any system described in the language must be acceptable as a subsystem of a system to be described in the language. SIMULA does not allow activity declarations within activities. The process decomposition is one level only in SIMULA. Therefore, design of any system using SIMULA will not proceed by component decomposition, but by a similar but less defined means. In terms of ALGOL, the design will be complete when all of the system software components exist as one or more ALGOL procedures. It was mentioned in an earlier chapter that if a system is run by a driver activity simulating the input, then it is still a simulator. It is proposed then that the design be considered complete upon the arrival at the stage when the only activity existing in the simulator is the driver activity, excluding the hardware structures.

4.2.2 Example

As an example of the use of SIMULA for designing a software system, consider the design of a Version II ILLIAC IV Operating System.

Version II programs will be executed on the B6500 in ALGOL. These programs are to govern the initiation of special "algorithms" or kernels, which are sets of instructions manipulating ILLIAC IV itself. These kernels will reside on the ILLIAC IV disk.

Version II will act in a multiprogramming capacity. Four or five jobs may be in the mix simultaneously. Whether or not Version II would be feasible depends on the question of what fraction of time the PE's are idle. Obviously, it is not feasible to invest the time and money to design and code a new operating system without knowing if the effort would produce a more efficient system in terms of percentage of PE idle time compared to the corresponding data when operating under the Version I Operating System.

However, the problem very nicely fits the design by simulation using SIMULA, as described in the preceding section. The objectives are to design the system for existing hardware and to code the final operating system in ALGOL. It is impractical to proceed any other way.

This design will be a design-study application. Initially the designers are faced with four or five very well defined hardware components, as far as their behavior goes, and a set of not-very-well defined components, even as far as behavior goes, representing the modules of the Version II Operating System. Design should proceed in three stages.

Stage I--Behavior Definition of Hardware Components

It is assumed that the behavior of the hardware components are known to a great extent. This is mandatory in that the design of the operating system will depend upon the output of each of these components with respect to various inputs. It is suggested that statistical distributions be formulated for the behavior of the various hardware components for use in the operating system design. This stage may very well be the most time consuming of the whole design since all of the decisions concerning the feasibility of the Version II Operating System are governed by the behavior of these hardware components. The designer should also determine tolerance values for his data based upon his estimation of the accuracy of the models of the hardware components.

Stage 2--Behavior Definition of Software Components

In the design of this system, behavior definition of the components is a major part of the design phase. (Note that in all previous discussion, the problem of defining the behavior of the major components of the system was glossed over and made to seem trivial. This was not meant to be implied.) However, behavior definition is different in this case in that the behavior must be defined to fit existing hardware whereas, in previous discussions, behavior definition was not restricted in this manner and extended to logic elements or to "sequential processes", the division just prior to deciding which is to be hardware and which is to be software. Using a simulator to define this behavior would promote a well integrated system as a whole and would alleviate the situation of getting well into the definition of several individual components only to find that they will not work together.

Stage 3--Refinement of the Code with Activities to Become Procedures.

Once the behavior of each component has reached an optimal definition as far as the hardware is concerned, it is a relatively simple matter to refine the code within the activity to the form in which it will exist in the final system. When all of the SIMULA constructs have been coded out of the system except for one lone driver activity, then this is the final form of the operating system.

5. SUMMARY

This study served three purposes: first, the design of a flexible simulator for examining certain aspects of the allocation of files for ILLIAC IV was discussed; second, the presentation of the philosophy of simulation and, finally, the presentation of a method of systems design.

The ILLIAC IV Disk File Allocator Simulator was constructed mainly for the purpose of providing a useful means of testing the performance of the Disk File Allocator. Another use of the simulator may be to do statistical studies on the interactions between components of the Operating System interfacing with the Disk File Allocator. Another criterion considered in the construction of the simulator was that of flexibility. Each activity and procedure can be extensively changed internally and the simulator can be recompiled fairly easily.

Inherent in the design of any simulation language is the definition of the concept of systems, and a description of the system in a simulation language constitutes the simulator itself minus only the "driver" for the simulator. It is hoped that this paper has presented the philosophies of systems and simulation and their interrelationships in such a way as to facilitate the design of any system simulator.

As systems in nearly every aspect of industry become more complex each year, the prospect of design by simulation offers intriguing possibilities.* This method of design would promote systems with built-in modularity. The top-down design approach should promote greater coordination between the modules or components of the system, resulting in a more efficient system

*The design, implementation, and analysis phases all take place concurrently.

with less errors in design once the final design stage is reached. Systems analysis and study should be taking place at every level of the design allowing finalization of many design details at a much earlier time than would be the case if simulation and a proper study of the system took place after system implementation.

APPENDIX A

INSTRUCTIONS FOR COMPILING THE SIMULATOR

Since the present SIMULA translator cannot handle constructs such as parameterized defines and case statements, both of which are used extensively in the disk file allocator and its array defines, the source code for the simulator exists in two parts: (1) the SIMULA translatable portion, consisting of the main SIMULA block and a few variables global to this block with dummy declarations for the untranslatable code, and (2) the non-SIMULA translatable portion, consisting of the disk file allocator, its procedures TREEBUILD and ALLOCATE, and its array declarations and defines.

The compilation process of this simulator takes place in three stages: (1) translation of the translatable code, (2) the merge of non-translatable code with the output of the SIMULA translation, and (3) the ALGOL compilation of the output of the merge.

The following are detailed instructions for compiling the simulator, assuming two files on disk; the translatable code, SIMULA/TI4DISK, and the untranslatable code, SIMULA/ALLOC, the disk file allocator patch deck file.

1. The execution of SIMULA/DISK will translate the SIMULA constructs into statements which can be compiled by the ALGOL compiler. In preparing the code for translation, precede each dummy section with "%BD", signifying the beginning of the section to be deleted, and end each of these sections with "%ED". Somewhere in the code between these two controls,

insert "%CP" <name of file to be inserted >, in this case, this file is SIMULA/ALLOC. (Note Appendix D.2).

The control cards for the translation are:

```
?USER=OPSYS
?EXECUTE SIMULA/DISK
?PRIORITY=3
?CORE=15000
?FILE TCODE=SIMULA/SI4DISK
?FILE DISK=SIMULA/TI4DISK
?FILE LINE=LINE BACK UP DISK
?FILE CARD=SIMDISK
?DATA SIMDISK
$DISK LIST
                                99999999 (in columns 73-80)
?END
```

If card input is used for the code to be translated, then the control cards are:

```
?USER=OPSYS
?EXECUTE SIMULA/DISK
?PRIORITY=3
?CORE=15000
?FILE TCODE=SIMULA/SI4DISK
?FILE LINE=LINE BACK UP DISK
?FILE CARD=SIMDISK
?DATA SIMDISK

      [source deck]

?END
```

(See Appendix D.1 for a sample of the translatable code, SIMULA/TI4DISK).

2. The file specified in %CP <file name> statement is now merged into the section enclosed by %BD and %ED. This merge is accomplished by executing SIMULA/MERGE, and outputs SIMULA/MI4DISK. The control cards are:

```
?USER=OPSYS
?EXECUTE SIMULA/MERGE
?FILE CARD=SIMULA/SI4DISK
?FILE NEWDECK=SIMULA/MI4DISK
?END
```

3. Finally the merged source code file SIMULA/MI4DISK is merged with SIMULA/GLOBAL, which contains the SIMULA procedures called in the translated code and the ALGOL compile is done, outputting the final executable code file, SIMULA/I4DISK. The control cards are:

```
?USER=OPSYS
?COMPILE SIMULA/I4DISK ALGOL LIBRARY
?ALGOL STACK=1000
?ALGOL FILE TAPE=SIMULA/GLOBAL DISK SERIAL
?ALGOL FILE CARE=SIMULA/MI4DISK DISK SERIAL
?END
```

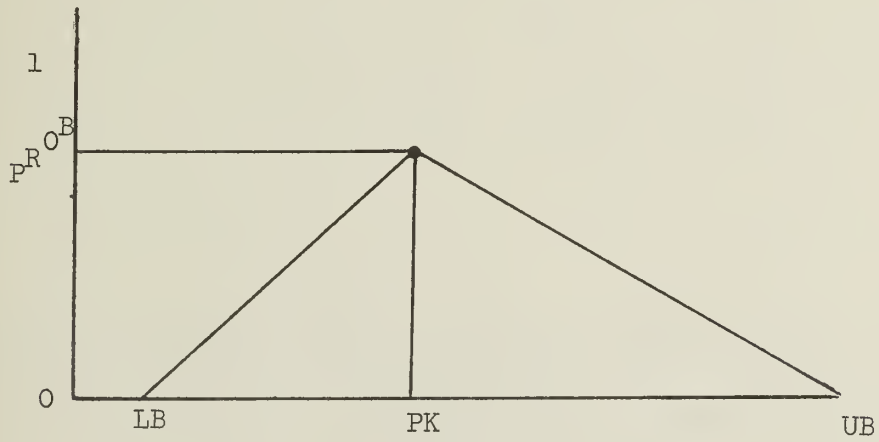
APPENDIX B

ADDITIONS TO THE B5500 IMPLEMENTATION OF SIMULA

B-1. PROCEDURE PEAKEDDRAW

This procedure allows the user to draw a number from a distribution of random numbers which is peaked about a specified number between two limits. The user specifies, as parameters to the procedure, the lower and upper limits, the value around which the numbers are peaked, and a random number drawn from a uniform distribution between 0 and 1.

The derivation of the peaked distribution was done in this way. Given a density function F peaked in the following way:

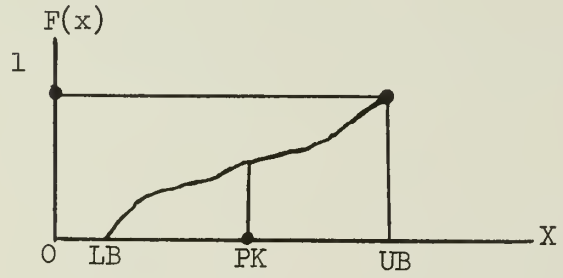


The distribution of function F is the following:

$$F(x) = \int_{-\infty}^{\infty} f(x)dx = \int_{LB}^{PK} \frac{PROB(X-LB)}{PK-LB}\,dx + \int_{PK}^{UB} \frac{PROB(UB-X)}{UB-PK}\,dx$$

Finding the value of PROB and then having an equation representing F will give us the means by which we can have a distribution function for drawing a peaked random number. We know from probability theory that $\int_{-\infty}^{\infty} f(x)\,dx = 1$ and this provides a means by which the value of PROB may be determined, given the lower bound, upper bound, and the value around which the numbers drawn are to be peaked.

Our final distribution function is then:



and the value of PK is $F^{-1}(U)$ where U is a random number drawn from a uniform number distribution between 0 and 1. After a certain amount of algebraic manipulation, the procedure is as follows:

```
REAL PROCEDURE PEAKEDDRAW (LB,UB,PEAK,UDISTR);
  VALUE LB,UB,PEAK,UDISTR; REAL LB,UB,PEAK,UDISTR;
    PEAKEDDRAW←IF UDISTRx(UB-LB)≤(PEAK-LB) THEN
      LB+SQRT(UDISTRx(PEAK-LB)x(UB-LB))
    ELSE
      UB-SQRT((UB-LB)x(UB-PEAK)x(1.0-UDISTR));
```

B-2. SIMULA/MERGE

The B5500 Implementation of SIMULA contains a program called SIMULA/PATCH which is supposed to allow the user to merge with ALGOL code with the translated code prior to ALGOL compilation.

However, SIMULA/PATCH does not work and the following program accomplishes the patching.

```
BEGIN
  COMMENT THIS PROGRAM WILL MERGE A DESIGNATED FILE WITH THE
  OUTPUT OF THE SIMULA TRANSLATOR, USING %.  THIS CAN BE USED WHEN
  PROGRAMS CONTAIN PARAMETERIZED DEFINES AND CASE STATEMENTS
  WHICH THE SIMULA TRANSLATOR CANNOT HANDLE.
  FOLLOWING ARE THE COMMANDS AND WHAT THEY DO.
          COMMAND          FUNCTION
         %BD              BEGIN DELETE AREA
         %ED              END DELETE AREA
         %CP P/S          COPY THE FILE P/S

  ;
  INTEGER INC,SEQMAX;
  FILE IN CARD DISK SERIAL (2,10,30);
  SAVE FILE OUT NEWDECK DISK SERIAL[20:450] (2,10,30,SAVE 99);
  ARRAY  A[0:9];
  POINTER P1,P72;
  INTEGER SEQ,LASTSEQ;
  ALPHA KMND;
  BOOLEAN DLTOG;  %TRUE IF CARD IMAGES ARE TO BE DELETED
  LABEL EOF1;
%
  DEFINE MAX(MAX1,MAX2)=
    (IF MAXA:=MAX1 GTR MAXB:=MAX2
          THEN MAXA ELSE MAXB)#;
  REAL MAXA,MAXB;
%
%
%
  PROCEDURE COPYIT;
    BEGIN
      FILE IN DISK DISK SERIAL (2,10,30);
      POINTER PP1;
      LABEL EOF;
      INTEGER I,K;
      ARRAY PRESUF[0:3];
      I:=0;
      PP1:=P1+3;
      THRU 2 DO
        BEGIN
          SCAN PP1:PP1 UNTIL IN ALPHA;
          REPLACE POINTER(PRESUF[I])+1
           BY  PP1:PP1 FOR K:7 WHILE IN
           ALPHA, " " FOR K;
          IF I=0 THEN
            BEGIN
              SCAN PP1:PP1 UNTIL = "/";
              PP1:=PP1+1;  %MOVE PAST /
              I:=2;
            END;  %IF
        END;  %THRU
      FILL DISK WITH PRESUF[0],PRFSUF[2];
      WHILE TRUE DO
        BEGIN
         READ (DISK,10,A[*])[EOF];
         REPLACE P72  BY  (LASTSEQ:=LASTSEQ+INC) FOR 8 DIGITS;
         WRITE (NEWDECK,10,A[*]);
        END;
EOF:
    END;  %COPYIT
%
```

```
%
%
  P1:=POINTER(A[0]);
  P72:=POINTER(A[9]);
  DLTOG:=FALSE;
  IF INC LEQ 0 THEN INC:=2;  %USER MAY SET INC USING COMMON
  SEQMAX:=26759900-INC-1;%LAST LOC BEFORE SIMULA/GLOBAL
  READ (CARD,10,A[*]);
  LASTSEQ:=SEQ:=INTEGER(P72,8);
  WHILE TRUE DO
    BEGIN
      COMMENT  LOOK FOR COMMANDS  ;
      KMND:=REAL(P1,3);
      IF KMND="%BD" THEN DLTOG:=TRUE ELSE
      IF KMND="%ED" THEN DLTOG:=FALSE ELSE
      IF KMND="%CP" THEN COPYIT
      ELSE IF NOT DLTOG THEN
       BEGIN
         IF INTEGER (P72,8) LEQ SEQMAX  THEN
         REPLACE P72  BY  (LASTSEQ:=MAX(SEQ,LASTSEQ+INC))
                                        FOR 8 DIGITS
         ELSE LASTSEQ:=SEQ;
         WRITE (NEWDECK,10,A[*]);
       END;
      READ (CARD,10,A[*])[EOF1];
      SEQ:=INTEGER(P72,8);
    END;
EOF1:
  LOCK(NEWDECK);
END.
```

B-3. PROCEDURE REPLACE

The present SIMULA translator is written in ALGOL. The following procedure allows the user to replace the XALGOL construct REPLACE POINTER (A[I]) + I1 BY POINTER (B[J]) + J1 FOR K WORDS by its ALGOL equivalent.

```
SIMULA /REPLACE LISTED AT  9:19 ON 69298 BY OPSYS

  STREAM PROCEDURE
  REPLACE(DESTARY,DESTOFFSET,SOURCEARY,SOURCEOFFSET,COUNT);
  VALUE  SOURCEOFFSET,DESTOFFSET,COUNT;
  %
  COMMENT REPLACE POINTER(A[I])+I1 BY POINTER(B[J])+J1 FOR K WORDS
   TRANSLATES TO:
   REPLACE (A[I],I1,B[J],J1,K);
%
  BEGIN
    LOCAL DIV64,MOD64;
    DI:=LOC DIV64;SI:=LOC COUNT;SI:=SI+6;DI:=DI+7;DS:=CHR;
    DI:=LOC MOD64;DI:=DI+7;DS:=CHR;
    SI:=SOURCEARY;SI:=SI+SOURCEOFFSET;
    DI:=DESTARY;DI:=DI+DESTOFFSET;
    DIV64(DS:=63 WDS;DS:=WDS);DS:=MOD64 WDS;
  END REPLACE;
```

APPENDIX C

A SIMPLE DISK FILE ALLOCATOR SIMULATOR

C-1. THE SOURCE CODE

This first section of APPENDIX C contains the source code for the simulator based on Knuth's [3] algorithms. It is assumed that there is a storage area of 20,000 units available for storage. The number of contiguous units requested is generated randomly and the allocator searches a storage directory, using the "first-fit" method, to find an available block containing an adequate number of contiguous units. When the allocated space is no longer needed, it is released as available space again.

```
SIMULA BEGIN

  INTEGER M;
  SET QUEUE;    %QUEUE FOR REQUESTS AWAITING ALLOCATION
  SET DOSSIER;   %SET OF REQUESTS SUCCESSFULLY ALLOCATED
  BOOLEAN NOSPACE;
  BOOLEAN ALLOCATORBUSY;
  ELEMENT X;
  ELEMENT ARRAY ALLOC[0:999];
  ELEMENT ARRAY QUE[0:999];
  INTEGER ARRAY WAITIME[0:999],
                RUNTIMES[0:250],
                   RUNTIME[0:999];
  DEFINE INDOSSIER=NUMINDOSSIER←CARDINAL(DOSSIER);
                    WRITE (LN,INDOS,NUMINDOSSIER)%;
  DEFINE INQUEUE=NUMINQUEUE←CARDINAL(QUEUE);
                   WRITE (LN,INQUE,NUMINQUEUE)%;
  INTEGER U;
  INTEGER ALOC,ASIZE;
  INTEGER NUMINQUEUE,NUMINDOSSIER;
  INTEGER P,Q,R;
  INTEGER T;
  INTEGER ARRAY FREELOC[0:999],FREESIZE[0:999],FREELINK[0:999];
  INTEGER ARRAY ALLOCNUM[0:999],ALLOCSIZE[0:999],ALLOCLOC[0:999];
  INTEGER LBDA,AVAIL;
  INTEGER U1,U2,U3,U4;
  REAL Z;
  ALPHA QUEUEHEAD;
  FILE BG 15(2,10);
  FILE LN 15(2,10);
  FORMAT TOP  (///X35,"D I S K     M A P "//X6,"POSITION IN DIRECTORY",
            X3,"FIRST FREE UNIT",X4,"SIZE OF BLOCK",X3,
               "DIRECTORY LINK"/),
         DISKMAP (X14,I4,X17,I6,X12,I8,X10,I8);
  FORMAT INDOS (//"THE NUMBER OF REQUESTS ALREADY ALLOCATED IS ",
                I6,"."),
          INQUE ("THE NUMBER OF REQUESTS IN THE QUEUE IS ",I6,"."/),
         RECREQ (//X6,"REQUEST",I5," HAS JUST COME IN FOR",I6,
           " CONTIGUOUS UNITS."/"TIME NOW IS ",I6,"."),
  ALLOCDONE (//X6,"REQUEST",I5," WAS JUST ALLOCATED AFTER WAITING ",
              I6," UNITS OF TIME"/
       "IN THE QUEUE.  ITS BEGINNING BLOCK LOCATION ON DISK IS",
        I6", AND"/"ITS BLOCK SIZE IS",I6,
            " UNITS. "/"TIME NOW IS ",I6,"."/),
         SPACEND (//X6,"WE HAVE NO SPACE ON DISK FOR REQUEST",I5,
         " IN THE QUEUE, SO WE TRY THE NEXT ELEMENT IN THE QUEUE."
           /"TIME NOW IS ",I6,"."),
        INTERIM (" "),
         RUNDONE (//X6,"REQUEST",I5," HAS FINISHED RUNNING AFTER ",I3
               /"UNITS OF TIME AND ITS SPACE IS RELEASED."
              /"TIME NOW IS ",I6,"."//);
  FORMAT F1 (//"B1"),
         F2 ("B2"),
         F3 ("B3"),
         F4 ("B4"),
         F5 ("B5"),
         F6 ("B6"//);
%
%
COMMENT PEAKEDDRAW DRAWS A RANDOM NUMBER FROM A DISTRIBUTION WHICH
IS PEAKED ABOUT A NUMBER BETWEEN A LOWER AND UPPER BOUND. ;
%
REAL PROCEDURE PEAKEDDRAW (LB,UB,PEAK,UDISTR);
```

```
   VALUE LB,UB,PEAK,UDISTR; REAL LB,UB,PEAK,UDISTR;
     PEAKFDDRAW←IF UDISTR×(UB-LB)≤(PEAK-LB) THEN
       LB+SQRT(UDISTR×(PEAK-LB)×(UB-LB))
     ELSE
       UB-SQRT((UB-LB)×(UB-PEAK)×(1.0-UDISTR));
%
%
COMMENT PROCEDURE ALLOCATOR USES THE AVAILABLE STORAGE MAP TO
SEARCH FOR SPACE IN ACCORDANCE WITH THE NUMBER OF CONTIGUOUS BLOCKS
REQUESTED FOR A SPECIFIC REQUEST. ;
%
  PROCEDURE SCHED ALLOCATOR(N);
  VALUE N;
    INTEGER N;
    BEGIN
      LABEL A1,A2,A3,A4;
      LABEL FIN;
      INTEGER K;
COMMENT A DIRECTORY OF AVAILABLE DISK SPACE - HAS THREE ENTRIES FOR EACH
AVAILABLE BLOCK OF UNITS,FREELOC,FREESIZE,AND FREELINK (POINTER TO NEXT
AVAILABLE BLOCK);
  A1: Q←AVAIL;
      IF FREESIZE[Q]≥N THEN
        BEGIN
          ALOC←FREELOC[Q];
          K←FREESIZE[Q]-N;
          IF K=0 THEN
              AVAIL←FREELINK[Q]
          ELSE
            BEGIN
              FREELOC[Q]←FREELOC[Q]+N;
              FREESIZE[Q]←K;
            END;
            GO TO FIN;
        END;
  A2: P←FREELINK[Q];
      IF P=LPDA THEN
        BEGIN
          NOSPACE←TRUE;
          GO TO FIN;
        END;
  A3: IF FREESIZE[P]≥N THEN
        BEGIN
          ALOC←FREELOC[P];
          K←FREESIZE[P]-N;
          IF K=0 THEN
              FREELINK[Q]←FREELINK[P]
        ELSE
          BEGIN
            FREELOC[P]←FREELOC[P]+N;
            FREESIZE[P]←K;
          END;
          GO TO FIN;
        END
      ELSE
        BEGIN
          Q←P;
          GO TO A2;
        END;
  FIN:
        HOLD (RANDINT(0,10,U3));
```

```
    END ALLOCATOR;
%
%
COMMENT PROCEDURE RELEASER RELEASES THE SPACE WHICH WAS ALLOCATED
TO A CERTAIN REQUEST WHEN IT IS NO LONGER NEEDED.;
%
  PROCEDURE RELEASER (PO,N);
  VALUE PO,N;
    INTEGER PO,N;
    BEGIN
      LABEL B1,B2,B3,B4,B5,B6;
      NOSPACE←FALSE;
  B1: R←AVAIL;
      IF FREELOC[AVAIL]>ALLOCLOC[PO] THEN
          GO TO B5;
  B2: P←FREELINK[R];
      IF P=LAMBDA THEN
          GO TO B3;
      IF FREELOC[P]>ALLOCLOC[PO] THEN
          GO TO B3
      ELSE
        BEGIN
          R←P;
          GO TO B2;
        END;
  B3: IF P≠LAMBDA THEN
          IF ALLOCLOC[PO]+N=FREELOC[P] THEN
            BEGIN
              N←N+FREESIZE[P];
              FREELINK[PO]←FREELINK[P];
            END
          ELSE
              FREELINK[PO]←P
      ELSE
          FREELINK[PO]←P;
  B4: IF FREELOC[R]+FREESIZE[R]=ALLOCLOC[PO] THEN
        BEGIN
          FREESIZE[R]←FREESIZE[R]+N;
          FREELINK[R]←FREELINK[PO];
        END
      ELSE
        BEGIN
          FREELINK[R]←PO;
          FREESIZE[PO]←N;
          FREELOC[PO]←ALLOCLOC[PO];
        END;
      GO TO B6;
  B5: IF ALLOCLOC[PO]+N=FREELOC[AVAIL] THEN
        BEGIN
          N←N+FREESIZE[AVAIL];
          FREELINK[PO]←FREELINK[AVAIL];
        END
      ELSE
          FREELINK[PO]←AVAIL;
      FREELOC[PO]←ALLOCLOC[PO];
      FREESIZE[PO]←N;
      AVAIL←PO;
  B6:
    END RELEASER;
%
%
```

```
COMMENT PROCEDURE PRINTDISKMAP PRINTS THE MAP OF AVAILABLE STORAGE
SPACE, GIVING THE ADDRESS OF THE INITIAL UNIT OF A BLOCK OF FREE
SPACE AND THE NUMBER OF CONTIGUOUS UNITS IN THAT BLOCK. ;
%
  PROCEDURE PRINTDISKMAP;
    BEGIN
      INTEGER J;
      J←AVAIL;
      WRITE (LN,TOP);
      WHILE J≠LBDA DO
        BEGIN
          WRITE (LN,DISKMAP,J,FREELOC[J],FREESIZE[J],FREELINK[J]);
          J←FREELINK[J];
        END;
    END PRINTDISKMAP;
%
  ACTIVITY JOBRUNNING (NUM,LOCATION,SIZE); INTEGER NUM,
    LOCATION,SIZE;FORWARD;
  ACTIVITY QUEUEREQUEST (REQFILES,REQNUM);INTEGER REQNUM,REQFILES;
    FORWARD;
%
%
COMMENT THE FOLLOWING ACTIVITY MAKES ALLOCATION ATTEMPTS ON EACH
SUCCESSIVE MEMBER OF THE QUEUE UNTIL IT IS EITHER SUCCESSFUL, IN WHICH
CASE IT ACTIVATES JOBRUNNING TO SIMULATE THE EXECUTION OF THE JOB TO
WHICH THE SPACE WAS ALLOCATED, OR UNSUCCESSFUL IN ALLOCATING ANY
MEMBER OF THE QUEUE, IN WHICH CASE THIS ACTIVITY TERMINATES.;
%
  ACTIVITY SUPERVISOR;
    BEGIN
      LABEL NEXTELEMENT;
      INTEGER J;
      IF NOT (ALLOCATORBUSY OR NOSPACE) THEN
        BEGIN
        ALLOCATORBUSY←TRUE;
        X←FIRST(QUEUE);
        IF NOT EMPTY(QUEUE) THEN
NEXTELEMENT;
          INSPECT X WHEN QUEUEREQUEST DO
            BEGIN
              J←REQNUM;
              ALLOCATOR (REQFILES);  %MAKE AN ALLOCATION ATTEMPT
              IF NOT NOSPACE THEN
                BEGIN
                  WAITIME[REQNUM]←SIMTIME-WAITIME[REQNUM];
                  ALLOCLOC[REQNUM]←ALOC;
                  ALLOCSIZE[REQNUM]←REQFILES;
                  WRITE (LN,ALLOCDONE,REQNUM,WAITIME[REQNUM],
                   ALOC,REQFILES,SIMTIME);
PRINTDISKMAP;
                  REMOVE (QUE[REQNUM]);
                  ALLOCATORBUSY←FALSE;
                  ACTIVATE (ALLOC[REQNUM]←NEW JOBRUNNING(REQNUM,
                   ALLOCLOC[REQNUM],ALLOCSIZE[REQNUM]));
                END;
            END;
          IF NOSPACE THEN
                IF X≠LAST(QUEUE) THEN
                  BEGIN
                    WRITE (LN,SPACEND,J,SIMTIME);
                    NOSPACE←FALSE;
```

```
                    X←SUC(X);
                    GO TO NEXTELEMENT;
                  END;
          ALLOCATORBUSY←FALSE;
          NOSPACE←FALSE;
        END;
    END SUPERVISOR;
%
%
COMMENT ACTIVITY JOBRUNNING SIMULATES THE EXECUTION OF THE JOB TO WHICH
THE SPACE WAS ALLOCATED.  THIS REPRESENTS THE TIME A CERTAIN
BLOCK IS UNAVAILABLE FOR REALLOCATION TO ANOTHER REQUEST.;
%
  ACTIVITY JOBRUNNING (NUM,LOCATION,SIZE);
    INTEGER NUM,LOCATION,SIZE;
    BEGIN
      INCLUDE (QUE[NUM],DOSSIER);  %AND PUT IN DOSSIER OF RUNNING JOBS
      INDOSSIER;
      INQUEUE;
      RUNTIME[NUM]←SIMTIME;
      Z←UNIFORM(0,1,U2);
      HOLD (PEAKFDDRAW(1,250,100,Z));
      RUNTIME[NUM]←SIMTIME-RUNTIME[NUM];
      T←RUNTIME[NUM];
      RUNTIMES[T]←RUNTIMES[T]+1;
      REMOVE (QUE[NUM]);
      RELEASER (NUM,SIZE);
      WRITE (LN,RUNDONE,NUM,RUNTIME[NUM],SIMTIME);
PRINTDISKMAP;
      ACTIVATE NEW SUPERVISOR;
    END JOBRUNNING;
%
%
COMMENT ACTIVITY QUEUEREQUEST PUTS THE NEW REQUEST INTO THE QUEUE
OF JOBS WAITING TO BE ALLOCATED AND THEN ACTIVIATES THE SUPERVISOR
TO CONTROL THE ALLOCATION ATTEMPTS.;
%
  ACTIVITY QUEUEREQUEST(REQFILES,REQNUM);
    INTEGER REQFILES;
    INTEGER REQNUM;
    BEGIN
      WAITIME[REQNUM]←SIMTIME;
      INCLUDE (QUE[REQNUM],CUEUE);
      WRITE (LN,RECREQ,REQNUM,REQFILES,SIMTIME);
      INDOSSIER;
      INQUEUE;
      ACTIVATE NEW SUPERVISOR;
    END QUEUEREQUEST;
%
%
COMMENT ACTIVITY GENERATEREQUEST GENERATES A RANDOM NUMBER BETWEEN
1 AND 2000 WHICH SIMULATES THE NUMBER OF CONTIGUOUS BLOCKS REQUESTED
BY A REQUEST.  REQUESTS ARE IDENTIFIED THROUGHOUT THE PROGRAM BY
A NUMBER WHICH IS ASSIGNED TO THEM IN THIS ACTIVITY.;
%
  ACTIVITY GENERATEREQUEST;
    BEGIN
      INTEGER I;
      LABEL LOOP;
      I←1;
LOOP;
```

```
        ACTIVATE (QUE[I]←NEW QUEUEREQUEST(RANDINT(1,2000,U1),I))
         DELAY 0;
        I←I+1;
        HOLD (RANDINT(1,20,U4));
        IF I≤99 THEN
            GO TO LOOP;
        BARGRAPH (BG,RUNTIMES[*],40,"R");
      END GENERATEREQUEST;
%
%
%  M A I N     P R O G R A M
    AVAIL←0;
    LBDA←20000;
    FREESIZE[0]←LBDA;
    FREELOC[0]←0;
    FREELINK[0]←LBDA;
    U1←3257;
    U2←46857;
    U3←94823;
    U4←45789;
    ACTIVATE NEW GENERATEREQUEST DELAY 0;
    CANCEL (CURRENT);
END SIMULA.
```

C-2. A SAMPLE OF THE EXECUTION OUTPUT

The execution output is self-explanatory. The time and the size of the block requested is noted each time a new request is generated. If a request is successfully allocated, this fact is noted on the output along with the request number, the units of simulation time that the request waited in the queue, the beginning location of the allocated space, the size of the block, and the current simulation time.

After storage is allocated and after storage is released, a free space map is printed of the available blocks of contiguous locations, indicated by the address of the first unit of the block and the size in units of the block.

When a run has been completed and the space is again released for allocation, this fact is noted on the output along with the request number, the simulated time of execution, and the current system time.

Another feature of SIMULA is the bargraph procedure in which a bargraph of a generated distribution is printed. In this example, bargraphs were made of the waiting times in the queue (WAITIME) and the running time (RUNTIME).

```
      REQUEST     1 HAS JUST COME IN FOR    795 CONTIGUOUS UNITS.
TIME NOW IS       0.


THE NUMBER OF REQUESTS ALREADY ALLOCATED IS       0.
THE NUMBER OF REQUESTS IN THE QUEUE IS       1.


      REQUEST     1 WAS JUST ALLOCATED AFTER WAITING       6 UNITS OF TIME
IN THE QUEUE.  ITS BEGINNING BLOCK LOCATION ON DISK IS      0, AND
ITS BLOCK SIZE IS    795 UNITS.
TIME NOW IS       6.


                                    D I S K     M A P

      POSITION IN DIRECTORY    FIRST FREE UNIT     SIZE OF BLOCK    DIRECTORY LINK

                  0                   795               19205            20000


THE NUMBER OF REQUESTS ALREADY ALLOCATED IS       1.
THE NUMBER OF REQUESTS IN THE QUEUE IS       0.


      REQUEST     2 HAS JUST COME IN FOR   1230 CONTIGUOUS UNITS.
TIME NOW IS      12.


THE NUMBER OF REQUESTS ALREADY ALLOCATED IS       1.
THE NUMBER OF REQUESTS IN THE QUEUE IS       1.


      REQUEST     2 WAS JUST ALLOCATED AFTER WAITING       6 UNITS OF TIME
IN THE QUEUE.  ITS BEGINNING BLOCK LOCATION ON DISK IS    795, AND
ITS BLOCK SIZE IS   1230 UNITS.
TIME NOW IS      18.


                                    D I S K     M A P

      POSITION IN DIRECTORY    FIRST FREE UNIT     SIZE OF BLOCK    DIRECTORY LINK

                  0                  2025               17975            20000


THE NUMBER OF REQUESTS ALREADY ALLOCATED IS       2.
THE NUMBER OF REQUESTS IN THE QUEUE IS       0.


      REQUEST     3 HAS JUST COME IN FOR   1467 CONTIGUOUS UNITS.
TIME NOW IS      22.
```

```
THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      2.
THE NUMBER OF REQUESTS IN THE QUEUE IS      1.


      REQUEST    3 WAS JUST ALLOCATED AFTER WAITING      5 UNITS OF TIME
IN THE QUEUE.  ITS BEGINNING BLOCK LOCATION ON DISK IS  2025, AND
ITS BLOCK SIZE IS  1467 UNITS.
TIME NOW IS     27.


                                D I S K    M A P

      POSITION IN DIRECTORY   FIRST FREE UNIT    SIZE OF BLOCK   DIRECTORY LINK

                  0                 3492              16508            20000


THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      3.
THE NUMBER OF REQUESTS IN THE QUEUE IS      0.


      REQUEST    4 HAS JUST COME IN FOR   130 CONTIGUOUS UNITS.
TIME NOW IS     40.


THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      3.
THE NUMBER OF REQUESTS IN THE QUEUE IS      1.


      REQUEST    4 WAS JUST ALLOCATED AFTER WAITING      0 UNITS OF TIME
IN THE QUEUE.  ITS BEGINNING BLOCK LOCATION ON DISK IS  3492, AND
ITS BLOCK SIZE IS   130 UNITS.
TIME NOW IS     40.


                                D I S K    M A P

      POSITION IN DIRECTORY   FIRST FREE UNIT    SIZE OF BLOCK   DIRECTORY LINK

                  0                 3622              16378            20000


THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      4.
THE NUMBER OF REQUESTS IN THE QUEUE IS      0.


      REQUEST    5 HAS JUST COME IN FOR    19 CONTIGUOUS UNITS.
TIME NOW IS     48.


THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      4.
```

```
THE NUMBER OF REQUESTS IN THE QUEUE IS      1.


      REQUEST    5 WAS JUST ALLOCATED AFTER WAITING      6 UNITS OF TIME
IN THE QUEUE.  ITS BEGINNING BLOCK LOCATION ON DISK IS  3622, AND
ITS BLOCK SIZE IS    19 UNITS.
TIME NOW IS     54.


                                D I S K    M A P

      POSITION IN DIRECTORY   FIRST FREE UNIT    SIZE OF BLOCK   DIRECTORY LINK

                  0                 3641              16359           20000


THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      5.
THE NUMBER OF REQUESTS IN THE QUEUE IS      0.


      REQUEST    6 HAS JUST COME IN FOR   719 CONTIGUOUS UNITS.
TIME NOW IS     60.

THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      5.
THE NUMBER OF REQUESTS IN THE QUEUE IS      1.


      REQUEST    7 HAS JUST COME IN FOR  1520 CONTIGUOUS UNITS.
TIME NOW IS     61.

THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      5.
THE NUMBER OF REQUESTS IN THE QUEUE IS      2.


      REQUEST    6 WAS JUST ALLOCATED AFTER WAITING      6 UNITS OF TIME
IN THE QUEUE.  ITS BEGINNING BLOCK LOCATION ON DISK IS  3641, AND
ITS BLOCK SIZE IS   719 UNITS.
TIME NOW IS     66.


                                D I S K    M A P

      POSITION IN DIRECTORY   FIRST FREE UNIT    SIZE OF BLOCK   DIRECTORY LINK

                  0                 4360              15680           20000


THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      6.
THE NUMBER OF REQUESTS IN THE QUEUE IS      1.
```

```
      REQUEST    8 HAS JUST COME IN FOR   413 CONTIGUOUS UNITS.
TIME NOW IS     74.


THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      6.
THE NUMBER OF REQUESTS IN THE QUEUE IS      2.


      REQUEST    7 WAS JUST ALLOCATED AFTER WAITING     18 UNITS OF TIME
IN THE QUEUE.  ITS BEGINNING BLOCK LOCATION ON DISK IS  4360, AND
ITS BLOCK SIZE IS  1520 UNITS.
TIME NOW IS     79.


                                D I S K    M A P

      POSITION IN DIRECTORY   FIRST FREE UNIT    SIZE OF BLOCK   DIRECTORY LINK

                  0                  5880              14120            20000


THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      7.
THE NUMBER OF REQUESTS IN THE QUEUE IS      1.


      REQUEST    9 HAS JUST COME IN FOR  1848 CONTIGUOUS UNITS.
TIME NOW IS     94.


THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      7.
THE NUMBER OF REQUESTS IN THE QUEUE IS      2.


      REQUEST    8 WAS JUST ALLOCATED AFTER WAITING     28 UNITS OF TIME
IN THE QUEUE.  ITS BEGINNING BLOCK LOCATION ON DISK IS  5880, AND
ITS BLOCK SIZE IS   413 UNITS.
TIME NOW IS    102.


                                D I S K    M A P

      POSITION IN DIRECTORY   FIRST FREE UNIT    SIZE OF BLOCK   DIRECTORY LINK

                  0                  6293              13707            20000


THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      8.
THE NUMBER OF REQUESTS IN THE QUEUE IS      1.


      REQUEST   10 HAS JUST COME IN FOR  1205 CONTIGUOUS UNITS.
TIME NOW IS    107.
```

```
THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      8.
THE NUMBER OF REQUESTS IN THE QUEUE IS      2.


      REQUEST    9 WAS JUST ALLOCATED AFTER WAITING     20 UNITS OF TIME
IN THE QUEUE.  ITS BEGINNING BLOCK LOCATION ON DISK IS  6293, AND
ITS BLOCK SIZE IS  1848 UNITS.
TIME NOW IS    114.


                                D I S K    M A P

      POSITION IN DIRECTORY   FIRST FREE UNIT    SIZE OF BLOCK   DIRECTORY LINK

                  0                 8141              11859            20000


THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      9.
THE NUMBER OF REQUESTS IN THE QUEUE IS      1.


      REQUEST   11 HAS JUST COME IN FOR   144 CONTIGUOUS UNITS.
TIME NOW IS    116.

THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      9.
THE NUMBER OF REQUESTS IN THE QUEUE IS      2.


      REQUEST   10 WAS JUST ALLOCATED AFTER WAITING     17 UNITS OF TIME
IN THE QUEUE.  ITS BEGINNING BLOCK LOCATION ON DISK IS  8141, AND
ITS BLOCK SIZE IS  1205 UNITS.
TIME NOW IS    124.


                                D I S K    M A P

      POSITION IN DIRECTORY   FIRST FREE UNIT    SIZE OF BLOCK   DIRECTORY LINK

                  0                 9346              10654            20000


THE NUMBER OF REQUESTS ALREADY ALLOCATED IS     10.
THE NUMBER OF REQUESTS IN THE QUEUE IS      1.


      REQUEST    4 HAS FINISHED RUNNING AFTER  89
UNITS OF TIME AND ITS SPACE IS RELEASED.
TIME NOW IS    129.
```

D I S K M A P

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
4	3492	130	0
0	9346	10654	20000

REQUEST 3 HAS FINISHED RUNNING AFTER 102
UNITS OF TIME AND ITS SPACE IS RELEASED.
TIME NOW IS 129.

D I S K M A P

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
3	2025	1597	0
0	9490	10510	20000

REQUEST 11 WAS JUST ALLOCATED AFTER WAITING 15 UNITS OF TIME
IN THE QUEUE. ITS BEGINNING BLOCK LOCATION ON DISK IS 9346, AND
ITS BLOCK SIZE IS 144 UNITS.
TIME NOW IS 131.

D I S K M A P

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
3	2025	1597	0
0	9490	10510	20000

THE NUMBER OF REQUESTS ALREADY ALLOCATED IS 0.
THE NUMBER OF REQUESTS IN THE QUEUE IS 0.

REQUEST 12 HAS JUST COME IN FOR 295 CONTIGUOUS UNITS.
TIME NOW IS 132.

THE NUMBER OF REQUESTS ALREADY ALLOCATED IS 0.
THE NUMBER OF REQUESTS IN THE QUEUE IS 1.

REQUEST 12 WAS JUST ALLOCATED AFTER WAITING 10 UNITS OF TIME
IN THE QUEUE. ITS BEGINNING BLOCK LOCATION ON DISK IS 2025, AND
ITS BLOCK SIZE IS 295 UNITS.
TIME NOW IS 142.

D I S K M A P

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
82	208	543	81
81	717	413	87
87	923	835	79
79	1926	5643	48
48	10399	9601	67
67	18386	1378	20000

REQUEST 96 HAS JUST COME IN FOR 1360 CONTIGUOUS UNITs.
TIME NOW IS 1075.

THE NUMBER OF REQUESTS ALREADY ALLOCATED IS 7.
THE NUMBER OF REQUESTS IN THE QUEUE IS 1.

REQUEST 95 HAS FINISHED RUNNING AFTER 17
UNITS OF TIME AND ITS SPACE IS RELEASED.
TIME NOW IS 1079.

D I S K M A P

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
82	208	543	81
81	717	413	87
87	923	835	95
95	1130	796	79
79	3286	4283	48
48	10399	9601	67
67	18386	1378	20000

REQUEST 96 WAS JUST ALLOCATED AFTER WAITING 5 UNITS OF TIME
IN THE QUEUE. ITS BEGINNING BLOCK LOCATION ON DISK IS 1926. AND
ITS BLOCK SIZE IS 1360 UNITS.
TIME NOW IS 1080.

D I S K M A P

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
82	208	543	81
81	717	413	87
87	923	835	95
95	1130	796	79
79	3286	4283	48
48	10399	9601	67
67	18386	1378	20000

```
THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      7.
THE NUMBER OF REQUESTS IN THE QUEUE IS      0.


      REQUEST   97 HAS JUST COME IN FOR   685 CONTIGUOUS UNITS.
TIME NOW IS   1093.

THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      7.
THE NUMBER OF REQUESTS IN THE QUEUE IS      1.


      REQUEST   83 HAS FINISHED RUNNING AFTER 163
UNITS OF TIME AND ITS SPACE IS RELEASED.
TIME NOW IS   1093.
```

D I S K M A P

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
82	208	543	81
81	717	413	83
83	751	172	87
87	1608	150	95
95	1130	796	79
79	3286	4283	48
48	10399	9601	67
67	18386	1378	20000

```
      REQUEST   97 WAS JUST ALLOCATED AFTER WAITING      1 UNITS OF TIME
IN THE QUEUE.  ITS BEGINNING BLOCK LOCATION ON DISK IS   923, AND
ITS BLOCK SIZE IS   685 UNITS.
TIME NOW IS   1094.
```

D I S K M A P

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
82	208	543	81
81	717	413	83
83	751	172	87
87	1608	150	95
95	1130	796	79
79	3286	4283	48
48	10399	9601	67
67	18386	1378	20000

```
THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      7.
```

THE NUMBER OF REQUESTS IN THE QUEUE IS 0.

REQUEST 98 HAS JUST COME IN FOR 1657 CONTIGUOUS UNITS.
TIME NOW IS 1112.

THE NUMBER OF REQUESTS ALREADY ALLOCATED IS 7.
THE NUMBER OF REQUESTS IN THE QUEUE IS 1.

REQUEST 92 HAS FINISHED RUNNING AFTER 78
UNITS OF TIME AND ITS SPACE IS RELEASED.
TIME NOW IS 1115.

D I S K M A P

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
92	0	499	82
82	208	543	81
81	717	413	83
83	751	172	87
87	1608	150	95
95	1130	796	79
79	4943	2626	48
48	10399	9601	67
67	18386	1378	20000

REQUEST 98 WAS JUST ALLOCATED AFTER WAITING 9 UNITS OF TIME
IN THE QUEUE. ITS BEGINNING BLOCK LOCATION ON DISK IS 3286, AND
ITS BLOCK SIZE IS 1657 UNITS.
TIME NOW IS 1121.

D I S K M A P

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
92	0	499	82
82	208	543	81
81	717	413	83
83	751	172	87
87	1608	150	95
95	1130	796	79
79	4943	2626	48
48	10399	9601	67
67	18386	1378	20000

THE NUMBER OF REQUESTS ALREADY ALLOCATED IS 7.
THE NUMBER OF REQUESTS IN THE QUEUE IS 0.

REQUEST 99 HAS JUST COME IN FOR 1907 CONTIGUOUS UNITS.
TIME NOW IS 1126.

THE NUMBER OF REQUESTS ALREADY ALLOCATED IS 7.
THE NUMBER OF REQUESTS IN THE QUEUE IS 1.

REQUEST 99 WAS JUST ALLOCATED AFTER WAITING 4 UNITS OF TIME
IN THE QUEUE. ITS BEGINNING BLOCK LOCATION ON DISK IS 4943, AND
ITS BLOCK SIZE IS 1907 UNITS.
TIME NOW IS 1130.

DISK MAP

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
92	0	499	82
82	208	543	81
81	717	413	83
83	751	172	87
87	1608	150	95
95	1130	796	79
79	6850	719	48
48	10399	9601	67
67	18386	1378	20000

THE NUMBER OF REQUESTS ALREADY ALLOCATED IS 8.
THE NUMBER OF REQUESTS IN THE QUEUE IS 0.

REQUEST 86 HAS FINISHED RUNNING AFTER 163
UNITS OF TIME AND ITS SPACE IS RELEASED.
TIME NOW IS 1136.

DISK MAP

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
92	0	499	82
82	208	543	81
81	717	413	83
83	751	172	87
87	1608	150	95
95	1130	796	79
79	6850	1905	48
48	10399	9601	67
67	18386	1378	20000

REQUEST 94 HAS FINISHED RUNNING AFTER 83
UNITS OF TIME AND ITS SPACE IS RELEASED.
TIME NOW IS 1143.

D I S K M A P

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
92	0	499	94
94	0	751	81
81	717	413	83
83	751	172	87
87	1608	150	95
95	1130	796	79
79	6850	1905	48
48	10399	9601	67
67	18386	1378	20000

REQUEST 91 HAS FINISHED RUNNING AFTER 151
UNITS OF TIME AND ITS SPACE IS RELEASED.
TIME NOW IS 1173.

D I S K M A P

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
92	0	499	94
94	0	751	81
81	717	413	83
83	751	172	87
87	1608	150	95
95	1130	796	79
79	6850	13150	67
67	18386	1378	20000

REQUEST 99 HAS FINISHED RUNNING AFTER 55
UNITS OF TIME AND ITS SPACE IS RELEASED.
TIME NOW IS 1185.

D I S K M A P

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
92	0	499	94
94	0	751	81

81	717	413	83
83	751	172	87
87	1608	150	95
95	1130	796	99
99	4943	15057	67
67	18386	1378	20000

REQUEST 96 HAS FINISHED RUNNING AFTER 112
UNITS OF TIME AND ITS SPACE IS RELEASED.
TIME NOW IS 1192.

D I S K M A P

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
92	0	499	94
94	0	751	81
81	717	413	83
83	751	172	87
87	1608	150	95
95	1130	2156	99
99	4943	15057	67
67	18386	1378	20000

REQUEST 93 HAS FINISHED RUNNING AFTER 198
UNITS OF TIME AND ITS SPACE IS RELEASED.
TIME NOW IS 1248.

D I S K M A P

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
92	0	499	94
94	0	751	93
93	499	631	83
83	751	172	87
87	1608	150	95
95	1130	2156	99
99	4943	15057	67
67	18386	1378	20000

REQUEST 98 HAS FINISHED RUNNING AFTER 137
UNITS OF TIME AND ITS SPACE IS RELEASED.
TIME NOW IS 1258.

D I S K M A P

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
92	0	499	94
94	0	751	93
93	499	631	83
83	751	172	87
87	1608	150	95
95	1130	18870	67
67	18386	1378	20000

REQUEST 97 HAS FINISHED RUNNING AFTER 235
UNITS OF TIME AND ITS SPACE IS RELEASED.
TIME NOW IS 1329.

D I S K M A P

POSITION IN DIRECTORY	FIRST FREE UNIT	SIZE OF BLOCK	DIRECTORY LINK
92	0	499	94
94	0	751	93
93	499	631	83
83	751	1007	95
95	1130	18870	67
67	18386	1378	20000

APPENDIX D

THE ILLIAC IV DISK FILE ALLOCATOR SIMULATOR

D-1. THE SIMULA TRANSLATABLE PORTION -- The Simulator Skeleton (SIMULA/TI4DISK).

The first section of Appendix D contains the SIMULA translatable portion of the simulator including the SIMULA constructs, namely, the SIMULA block containing the element and set declarations and the activity declarations.

As stated previously, the current version of the B5500 implementation of SIMULA will not accept parameterized defines or case statements; therefore, all code containing these constructs has been removed from this skeleton portion of the simulator before the SIMULA translation, but will be merged with the translated SIMULA code just prior to ALGOL compilation.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  SIMULA/TI4DISK--THE TRANSLATABLE PORTION
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
BEGIN
INTEGER I;
INTEGER QUEUENUM;
INTEGER ARRAY TIMESHARD[0:99];
  INTEGER T;
FILE CD(2,10);
FILE SIMFILE 15(2,15);
FORMAT FQUENUM (I2),
         FSHARDNUM (I3);
DEFINE IN=SIMFILE#;
%
COMMENT THE FOLLOWING PROCEDURE ALLOWS THE TRANSFER OF XALGOL
REPLACE STATEMENTS INTO EQUIVALENT ALGOL STATEMENTS;
%
  STREAM PROCEDURE
  REPLACE(DESTARY,DESTOFFSET,SOURCEARY,SOURCEOFFSET,COUNT);
  VALUE SOURCEOFFSET,DESTOFFSET,COUNT;
  %
  COMMENT REPLACE POINTER(A[I])+I1 BY POINTER(B[J])+J1 FOR K
   TRANSLATES TO:
   REPLACE (A[I],I1,B[J],J1,K);
%
  BEGIN
    LOCAL DIV64,MOD64;
    DI:=LOC DIV64;SI:=LOC COUNT;SI:=SI+6;DI:=DI+7;DS:=CHR;
    DI:=LOC MOD64;DI:=DI+7;DS:=CHR;
    SI:=SOURCEARY;SI:=SI+SOURCEOFFSET;
    DI:=DESTARY;DI:=DI+DESTOFFSET;
    DIV64(DS:=63 WDS;DS:=WDS);DS:=MOD64 WDS;
  END REPLACE;
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  BEGIN SIMULA/MERGE CONTROL SECTION
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%ED
  PROCEDURE TREEBUILD;  %DUMMY PROCEDURE
    BEGIN
    END;
%
%
  PROCEDURE ALLOCATE;  %DUMMY PROCEDURE
    BEGIN
    END;
%CP SIMULA/ALLOC
%
  PROCEDURE READEM;  %DUMMY PROCEDURE
    BEGIN
    END;
%
%
```

```
  PROCEDURE WRITEM;  %DUMMY PROCEDURE
    BEGIN
    END;
%
%ED
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  END SIMULA/MERGE CONTROL SECTION
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
READ (CD,FQUENUM,QUEUENUM);
QUEUENUM:=QUEUENUM-1;   %NUMBER OF QUEUES IN SYSTEM
FOR I←0 STEP 1 UNTIL QUEUENUM DO
    READ (CD,FSHARDNUM,TIMESHARD[I]);
%
%
COMMENT THIS SECTION BEGINS THE SIMULA BLOCK.  ALL SIMULA CON-
STRUCTS, ELEMENTS, SETS, AND ACTIVITIES ARE DEFINED IN THIS
BLOCK.  NO PARAMETERIZED DEFINES OR CASE STATEMENTS MAY OCCUR HERE.;
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
SIMULA BEGIN
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  INTEGER M;
  SET ARRAY QUEUE[0:QUEUENUM];     %REQUEST QUEUE
  SET DOSSIER;    %SET OF REQUESTS SUCCESSFULLY ALLOCATED
  BOOLEAN NOSPACE;
  BOOLEAN ALLOCATORBUSY;
  ELEMENT X;
  ELEMENT ARRAY QUE[0:999];
  INTEGER ARRAY WAITIME[0:999],
                RUNTIME[0:999],
               TIMESLICE[0:QUEUENUM],
                TREEBUILDTIME[0:999],
                ALLOCTIME[0:999];
  DEFINE NEQ=≠#,
         LSS=<#,
         GTR=>#,
         LEQ=≤#,
         GEQ=≥#;
%
% THE FIRST TWO DEFINES GIVE THE NUMBER OF ITEMS IN THE SET
% DOSSIER AND QUEUE[M]
%
  DEFINE INDOSSIER=NUMINDOSSIER←CARDINAL(DOSSIER);
                   WRITE (LN,INDOS,NUMINDOSSIER)#,
         INQUEUES=FOR M:=0 STEP 1 UNTIL QUEUENUM DO
              BEGIN
                NUMINQUEUE[M]←CARDINAL(QUEUE[M]);
                WRITE (LN,INQUE,M,NUMINQUEUE[M]);
              END#,
%
% THIS PRINTS THE CURRENT VALUES OF THE TIMESHARD AND TIMESLICE
% FOR A PARTICULAR QUEUE
%
         QUEUEINFO=WRITE (LN,INTERIM);
                FOR M←0 STEP 1 UNTIL QUEUENUM DO
```

```
            WRITE (LN,FSLICE,M,TIMESHARD[M],TIMESLICE[M])#;
   INTEGER U1,U2,U3,U4,U5;
   INTEGER NUMINDOSSIER;
   INTEGER ARRAY NUMINQUEUE[0:99];
   INTEGER ARRAY FREELOC[0:999],FREESIZE[0:999],FRELINK[0:999];
   INTEGER ARRAY ALLOCNUM[0:999],ALLOCSIZE[0:999],ALLOCLOC[0:999];
   REAL Z;
   FORMAT INDOS ("SIM--THE NUMBER OF REQUESTS ALREADY ALLOCATED IS ",
                 I6,".")，
          INQUE ("SIM--THE NUMBER OF REQUESTS IN QUEUE[",I2,"] IS ",
                 I4,"."),
          FSLICE ("SIM--QUEUE[",I2,"]-",X5,"TIMESHARD IS ",I6,
              " UNITS.  TIMESLICE IS ",I6," UNITS."),
          FTB("SIM--REQUEST ",I5," - PROCESSOR TIME FOR TREEBUILD IS ",
                      I6,".///),
           FAT("SIM--REQUEST ",I5," -PROCESSOR TIME FOR ALLOCATION IS ",
              I8,"."),
          FORMIN(///"SIM--REQUEST ",I5," HAS JUST ENTERED QUEUE ",I3,
      " AT SIMULATION TIME ",I6,"."),
          FALLOC (///"SIM--REQUEST ",I5,
                " WAS JUST ALLOCATED AFTER WAITING ",I5,
                " UNITS OF SIMULATION TIME IN QUEUE ",I3/
          "SIM--SIMTIME NOW IS ",I6,"."),
          FSEARCH ("SIM--ALLOCATION ATTEMPT TO BE MADE ON QUEUE[",
                 I2,"]."),
          FTHRU (///"SIM--REQUEST ",I3," IS FINISHED AFTER ",
               I5," UNITS OF TIME."),
          INTERIM (" ");
%
%
  REAL PROCEDURE PEAKEDDRAW(LB,UB,PEAK,UDISTR);
VALUE LB,UB,PEAK,UDISTR;  REAL LB,UB,PEAK,UDISTR;
   PEAKEDDRAW ← IF UDISTR×(UB-LB) ≤ (PEAK-LB)  THEN
                  LB+SQRT(UDISTR×(PEAK-LB)×(UB-LB))
         ELSE
                  UB-SQRT( (UB-LB)×(UB-PEAK)×(1.0-UDISTR) );
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   SIMULATOR ACTIVITY DECLARATIONS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
  ACTIVITY JOBRUNNING (NUM,WHICHQUEUE);INTEGER NUM,WHICHQUEUE;
    FORWARD;
  ACTIVITY QUEUEREQUEST (REQNUM);INTEGER REQNUM;LOCALS
    INTEGER QUENUM;FORWARD;
%
%
COMMENT THE FOLLOWING ACTIVITY CHECKS EACH QUEUE AND ON THE BASIS
OF ITS TIMESLICE VALUE, EITHER SELECTS A JOB FROM THIS QUEUE OR
CONTINUES ON TO THE NEXT QUEUE.  IF THERE IS AN ALLOCATION
ATTEMPT ON A REQUEST WITHIN A QUEUE AND IT FAILS, THE NEXT
REQUEST IN THE SAME QUEUE IS TAKEN FOR THE NEXT ATTEMPT.;
%
%
  ACTIVITY SUPERVISOR;
    BEGIN
      LABEL CHECKQUEUES,NEXTELEMENT;
```

```
      INTEGER J;
      IF NOT ALLOCATORBUSY THEN
        BEGIN
          ALLOCATORBUSY←TRUE;
          J←0;
CHECKQUEUES;
          X←FIRST(QUEUE[J]);
          IF TIMESLICE[J]>0 AND NOT EMPTY (QUEUE[J]) THEN
            BEGIN
NEXTELEMENT;
              INSPECT X WHEN QUEUEREQUEST DO
                BEGIN
                  WRITE (LN,FSEARCH,J);
                  TREEBUILDTIME[REQNUM]:=TIME(2);
                  TREEBUILD;
                  TREEBUILDTIME[REQNUM]:=TIME(2)-TREEBUILDTIME
                   [REQNUM];
                  WRITE (LN,FTB,REQNUM,TREEBUILDTIME[REQNUM]);
                  WRITEM;
                  ALLOCTIME[REQNUM]:=TIME(2);
                  ALLOCATE;
                  ALLOCTIME[REQNUM]:=RANDINT(1,500,U4);
                  IF NOT NOSPACE THEN
                    BEGIN
                      WAITIME[REQNUM]:=SIMTIME-WAITIME[REQNUM];
                      WRITE (LN,FALLOC,REQNUM,WAITIME[REQNUM],
                       J,SIMTIME);
                      WRITE (LN,FAT,REQNUM,ALLOCTIME[REQNUM]);
                      REMOVE(QUE[REQNUM]);
                      ALLOCATORBUSY←FALSE;
                      ACTIVATE (QUE[REQNUM]:=NEW JOBRUNNING
                       (REQNUM,J));
                    END;
                END;
              IF NOSPACE THEN
                  IF X≠LAST(QUEUE[J]) THEN
                    BEGIN
                      X←SUC(X);
                      NOSPACE←FALSE;
                      GO TO NEXTELEMENT;
                    END
                  ELSE
                      ALLOCATORBUSY←FALSE;
            END
          ELSE
            BEGIN
              J←J+1;
              IF J≤QUEUENUM THEN
                  GO TO CHECKQUEUES
              ELSE
                BEGIN
                  FOR J←0 STEP 1 UNTIL QUEUENUM DO
                    BEGIN
                      TIMESLICE[J]←TIMESLICE[J]+TIMESHARD[J];
                      IF TIMESLICE[J]>0 THEN
                          TIMESLICE[J]←TIMESHARD[J];
                      ALLOCATORBUSY←FALSE;
                    END;
                  QUEUEINFO;
                END;
            END;
```

```
      END;
    END SUPERVISOR;
%
%
COMMENT THE FOLLOWING ACTIVITY SIMULATES THE HOLDING OF THE
ALLOCATED SPACE FOR A RANDOM LENGTH OF TIME.  THE NEWLY ALLOCATED
REQUEST IS PUT INTO THE DOSSIER OF RUNNING JOBS AND HELD THERE
UNTIL ITS TIME, AS INDICATED BY THE RANDOM NUMBER DRAWING
PROCEDURE, IS UP.  THE JOB IS THEN REMOVED FROM DOSSIER AND
THE SUPERVISOR IS ACTIVATED TO ALLOCATE ANOTHER REQUEST.;
%
%
  ACTIVITY JOBRUNNING(NUM,WHICHQUEUE);
    INTEGER NUM,WHICHQUEUE;
    BEGIN
      INCLUDE (QUE[NUM],DOSSIER);  %AND PUT IN DOSSIER OF RUNNING JOBS
      INDOSSIER;
      INQUEUES;
      RUNTIME[NUM]←SIMTIME;
      Z←UNIFORM(0,1,U3);
      HOLD (PEAKEDDRAW(5,250,100,Z));
      RUNTIME[NUM]←SIMTIME-RUNTIME[NUM];
      WRITE (LN,FTHPU,NUM,RUNTIME[NUM]);
      REMOVE (QUE[NUM]);
      TIMESLICE[WHICHQUEUE]←TIMESLICE[WHICHQUEUE]-RUNTIME[NUM];
      QUEUEINFO;
      ACTIVATE NEW SUPERVISOR;
    END JOBRUNNING;
%
%
COMMENT THE FOLLOWING ACTIVITY RANDOMLY CHOOSES A QUEUE FOR
EACH REQUEST AND ENTERS THE REQUEST IN THE QUEUE.;
%
%
  ACTIVITY QUEUEREQUEST(REQNUM);
    INTEGER REQNUM;
    BEGIN
      INTEGER QUENUM;
      WAITIME[REQNUM]←SIMTIME;
      IF QUEUENUM=0 THEN QUENUM:=0
      ELSE QUENUM:=RANDINT(0,QUEUENUM,U4);
      WRITE (LN,FORMIN,REQNUM,QUENUM,SIMTIME);
      INCLUDE (QUE[REQNUM],QUEUE[QUENUM]);
      INDOSSIER;
      INQUEUES;
      ACTIVATE NEW SUPERVISOR;
    END QUEUEREQUEST;
%
%
COMMENT THE FOLLOWING ACTIVITY WAITS FOR AN AMOUNT OF SIMULATION
TIME SPECIFIED BYARANDOM DRAWING FROM A PEAKED DISTRIBUTION,
GETS A REQUEST, AND THEN ACTIVATES ACTIVITY QUEUEREQUEST.;
%
%
  ACTIVITY GETREQUEST;
    BEGIN
      INTEGER I;
      LABEL LOOP;
      I←1;
      READEM;  %THIS WILL READ SAME REQUEST OVER & OVER
LOOP;
```

```
        ACTIVATE(QUE[I]:=NEW QUEUEREQUEST(I)) DELAY 0;
        I:=I+1;
        Z:=UNIFORM(0,1,U2);
        HOLD (PEAKEDDRAW(1,100,30,Z));
        IF I<99 THEN
            GO TO LOOP;
    END GETREQUEST;
%
%
%  M A I N     P R O G R A M
  U1:=54967;
  U2:=34875;
  U3:=4589;
  U4:=678943;
  ACTIVATE NEW GETREQUEST DELAY 0;
  CANCEL (CURRENT);
END SIMULA;
END.
```

D-2. THE SIMULA UNTRANSLATABLE PORTION -- THE DISK FILE ALLOCATOR AND TREEBUILD PROCEDURES, GLOBAL DECLARATIONS AND DEFINITIONS (SIMULA/ALLOC)

This section of Appendix D contains the procedure for the allocation of disk space for the ILLIAC IV disk and other procedures, such as Treebuild, and arrays which interface with the disk file allocator and the system.

The disk file allocator in this simulator is not logically debugged. It was coded hastily by the author from system flow charts and preliminary definitions made of its arrays in an effort to get a procedure having the same interfaces with the system as was planned for the final allocator. This was because coding on this aspect of the ILLIAC IV Operating System had not begun at the time the coding on the simulator was completed.

Two other procedures are contained in this section. PROCEDURE READEM reads the data and sets up a file block. This block (logical records of 30 words each) constitutes the input to PROCEDURE TREEBUILD. PROCEDURE WRITEM outputs the results of the Treebuild procedure. These two procedures, and TREEBUILD are the property of the ILLIAC IV Operating System group.

```
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  SIMULA/ALLOC--THE UNTRANSLATABLE PORTION
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% D I S K    F I L E    A L L O C A T O R -
%
%    A R R A Y S ,    D E F I N E S ,    A N D    P R O C E D U R E S .
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%
    INTEGER ARRAY ADDSEGS[0:3];
    INTEGER NN,NXTBLK,NXASSN,P,NXFIL,NEXT,LASTIN,AS;
    INTEGER FIRSTSUNUM,LASTSUNUM,FIRSTEUNUM,LASTEUNUM;
    ALPHA A,B;
    BOOLEAN TOP;
FILE INN (2,10);
    FILE PASSFILE DISK RANDOM [3:300] (2,30);
  INTEGER AMOUNTOFFREEDISK,NUMSEGSPERTRK;
  INTEGER TOPBLOCK;
  PROCEDURE FREEMAPSPACE(A);
    INTEGER A;
    BEGIN
      %WHAT IS THIS...
    END;
%
 PROCEDURE GETMAPSPACE(A);
    INTEGER A;
    BEGIN
      %WHAT IS THIS...
    END;
%
  PROCEDURE GETASSNBLKSPACE(A);
    INTEGER A;
    BEGIN
      %WHAT IS THIS...
    END;
%
%
%
%
%
%
% THE FOLLOWING ARE THE GLOBAL ARRAYS AND DEFINES USED BY THE DISK FILE
% ALLOCATOR PROCEDURES, TREEBUILD AND ALLOCATE, AND THEIR PROCEDURES.
%
% PROCEDURE TREEBUILD-CREATED ARRAYS ARE:
```

```
%
      ALPHA ARRAY THING[0:9];
%
       REAL ARRAY PTRBLK [0:63,0:191],
                  ASSNBLK [0:63,0:191],
                  TOPBLK[0:63,0:191],
                  FILBLK [0:63,0:191],
                  HOLDIT[0:29];
%
% TREFBUILD DEFINES ARE :
%
%
     DEFINE NUMBPTR(NUMBPTR1)=PTRBLK[NUMBPTR1.[36:6],3*NUMBPTR1.[42:6]].
             [ 1: 9]#,     %46=47.
            VPPTR(VPPTR1)=PTRBLK[VPPTR1.[36:6],3*VPPTR1.[42:6]].
             [10:1]#,
            NUMSEGS(NUMSEGS1)=PTRBLK[NUMSEGS1.[36:6],3*NUMSEGS1.[42:6]].
             [11:21]#,
            NUMFRAGS(NUMFRAGS1)=PTRBLK[NUMFRAGS1.[36:6],3*NUMFRAGS1.[42:6
    ]].[32:16]#,
            NXTPTR(NXTPTR1)=PTRBLK[NXTPTR1.[36:6],3*NXTPTR1.[42:6]+1].
             [1:11]#,
            PHYPTR(PHYPTR1)=PTRBLK[PHYPTR1.[36:6],3*PHYPTR1.[42:6]+1].
             [12:12]#,
            VSUPTR(VSUPTR1)=PTRBLK[VSUPTR1.[36:6],3*VSUPTR1.[42:6]+1].
             [24:12]#,
            VTRKPTR(VTRKPTR1)=PTRBLK[VTRKPTR1.[36:6],3*VTRKPTR1.[42:6]+1]
            .[36:12]#,
            PHASPTR(PHASPTR1)=PTRBLK[PHASPTR1.[36:6],3*PHASPTR1.[42:6]+2]
            .[ 1:15]#,     %46=47.
            CONTIGPTR(CONTIGPTR1)=PTRBLK[CONTIGPTR1.[36:6],3*CONTIGPTR1.
    [42:6]+2].[16:16]#,
            JUNKPTR(JUNKPTR1)=PTRBLK[JUNKPTR1.[36:6],3*JUNKPTR1.[42:6]+2]
            .[32:16]#,
            CONT = B.[7:1]#,
            PH = B.[8:1]#,
            VEU = B.[ 9:1]#,
            VSU = B.[10:1]#,
            VTRK = B.[11:1]#,
            PEU  = B.[12:1]#,
            PSU  = B.[13:1]#,
            PTRK = B.[14:1]#,
            EUNUM =B.[15:4]#,
            SUNUM =B.[19:8]#,
            TRKNUM=B.[27:10]#,
            PHS   =B.[37:11]#,
            SEGS  =A.[24:24]#,
            NXTFRAG(NXTFRAG1)=ASSNBLK[NXTFRAG1.[36:6],2*NXTFRAG1.[42:6]].
             [ 1:15]#,     %46=47.
            NXTBL(NXTBL1)=ASSNBLK[NXTBL1.[36:6],2*NXTBL1.[42:6]].
             [16:16]#,
            SEGNUM(SEGNUM1)=ASSNBLK[SEGNUM1.[36:6],2*SEGNUM1.[42:6]+1].
             [ 1:20]#,     %46=47.
            PHNUM(PHNUM1)=ASSNBLK[PHNUM1.[36:6],2*PHNUM1.[42:6]+1].
             [21:11]#,
            ASSNBLKEU(ASSNBLKEU1)=ASSNBLK[ASSNBLKEU1.[36:6],3*
             ASSNBLKEU1.[42:6]+2].[1:4]#,
            ASSNBLKSU(ASSNBLKSU1)=ASSNBLK[ASSNBLKSU1.[36:6],3*
             ASSNBLKSU1.[42:6]+2].[5:4]#,
            ASSNBLKTRK(ASSNBLKTRK1)=ASSNBLK[ASSNBLKTRK1.[36:6],3*
             ASSNBLKTRK1.[42:6]+2].[9:3]#,
```

```
            ASSNBLKSEG(ASSNBLK FG1)=ASSNBLK[ASSNBLKSEG1.[36:6],3x
             ASSNBLKSEG1.[42:6]+2],[12:11]#,
            SYSLINK = HOLDIT[0],[24:24]#,
            WDSRCD  = HOLDIT[0],[19:5]#,
            AFILNAME(AFILNAME1) = FILBLK[AFILNAME1,[36:6],6xAFILNAME1,
                [42:6]]#,
            BFILNAME(BFILNAME1) = FILBLK[BFILNAME1,[36:6],6xBFILNAME1,
                [42:6]+1]#,
            CFILNAME(CFILNAME1) = FILBLK[CFILNAME1,[36:6],6xCFILNAME1,
                [42:6]+2]#,
            SEGSRECD(SEGSRECD1) = FILBLK[SEGSRECD1,[36:6],6xSEGSRECD1,
                [42:6]+3],[1:23]#,       %46=47,
            TOTSEGS(TOTSEGS1) = FILBLK[TOTSEGS1,[36:6],6xTOTSEGS1,
                [42:6]+3],[24:24]#,
            PREPTR(PREPTR1) = FILBLK[PREPTR1,[36:6],6xPREPTR1,
                [42:6]+4],[1:23]#,         %46=47,
            POSTPTR(POSTPTR1) = FILBLK[POSTPTR1,[36:6],6xPOSTPTR1,
                [42:6]+4],[24:24]#,
            FILASSN(FILASSN1) = FILBLK[FILASSN1,[36:6],6xFILASSN1,[42:6]
                +5],[ 1:11]#,     %46=47,
            PHYSICALFU(PHYSICALFU1)=TOPBLK[PHYSICALFU1,[36:6],2x
             PHYSICALFU1,[42:6]],[1:9]#,
                    VIRTUALFU(VIRTUALFU1)=TOPBLK[VIRTUALFU1,[36:6],2x
             VIRTUALFU1,[42:6]],[10:12]#,
            PHYSICALSU(PHYSICALSU1)=TOPBLK[PHYSICALSU1,[36:6],2x
             PHYSICALSU1,[42:6]],[46:1]#,
            PHYSICALTRK(PHYSICALTRK1)=TOPBLK[PHYSICALTRK1,[36:6],2x
             PHYSICALTRK1,[42:6]],[47:1]#,
            VIRTUALSU(VIRTUALSU1)=TOPBLK[VIRTUALSU1,[36:6],2x
             VIRTUALSU1,[42:6]],[22:12]#,
            VIRTUALTRK(VIRTUALTRK1)=TOPBLK[VIRTUALTRK1,[36:6],2x
             VIRTUALTRK1,[42:6]],[34:12]#,
            PHASEDPTR(PHASEDPTR1)=TOPBLK[PHASEDPTR1,[36:6],2x
             PHASEDPTR1,[42:6]+1],[1:15]#,
            CONTGPTR(CONTGPTR1)=TOPBLK[CONTGPTR1,[36:6],2x
             CONTGPTR1,[42:6]+1],[16:16]#,
            JNKPTR(JNKPTR1)=TOPBLK[JNKPTR1,[36:6],2x
             JNKPTR1,[42:6]+1],[32:16]#,
            ASEGSRECD = HOLDITT  7],[ 1:23]#,    %46=47,
            ATOTSEGS  = HOLDITT  7],[24:24]#,
            APREPTR   = HOLDITT  8],[ 1:23]#,     %46=47,
            APOSTPTR  = HOLDITT  8],[24:24]#;
%
% PROCEDURE ALLOCATE-CREATED ARRAYS ARE:
%
      REAL ARRAY MAPBLKPTRARRAY[0:63,0:191],
                 MAPBLK[0:63,0:191],
                 MAPSEG[0:1022];
%
% ALLOCATE DEFINES ARE:
%
  DEFINE NUMSUFRAGS(NUMSUFRAGS1,NUMSUFRAGS2)=MAPBLKPTRARRAY[NUMSUFRAGS1,
   NUMSUFRAGS2],[1:12]#,
  NUMAVAILSUSEGS(NUMAVAILSUSEGS1,NUMAVAILSUSEGS2)=MAPBLKPTRARRAY
   [NUMAVAILSUSEGS1,NUMAVAILSUSEGS2],[13:13]#,
  WHICHMAPNUM(WHICHMAPNUM1,WHICHMAPNUM2)=MAPBLKPTRARRAY[WHICHMAPNUM1,
   WHICHMAPNUM2],[26:14]#,
        MFU(MFU1)=MAPBLK[MFU1,[36:6],MFU1,[42:6]],[1:4]#,
        MSU(MSU1)=MAPBLK[MSU1,[36:6],MSU1,[42:6]],[5:4]#,
        MLOCK(MLOCK1)=MAPBLK[MLOCK1,[36:6],MLOCK1,[42:6]],[9:1]#,
```

```
        NMAPFRAGMENTS(NMAPFRAGMENTS1)=MAPBLK[NMAPFRAGMENTS1.[36:6],
         NMAPFRAGMENTS1.[42:6]].[10:12]#,
        NUMSEGINMAP(NUMSEGINMAP1)=MAPBLK[NUMSEGINMAP1.[36:6],
         NUMSEGINMAP1.[42:6]].[22:13]#,
        MAPPTR(MAPPTR1)=MAPBLK[MAPPTR1.[36:6],MAPPTR1.[42:6]].
         [35:13]#,
        FIRSTSEG(FIRSTSEG1)=MAPSEG[FIRSTSEG1].[1:13]#,
        NUMSEG(NUMSEG1)=MAPSEG[NUMSEG1].[14:13]#,
        NEXTMAPBLOCK(NEXTMAPBLOCK1)=MAPSEG[NEXTMAPBLOCK1].[27:14]#;
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PROCEDURE TREEBUILD BUILDS THE ALLOCATOR TREE FROM THE DISK FILE
% BLOCK BUILT BY THE JOB PARSER.
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  PROCEDURE TREEBUILD;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
BEGIN
    INTEGER N,N1;
    LABEL LPSU,LPTRK,LVFU,LVSU,LVTRK,FIN3,FIN4,LP9,RD;
    SWITCH LNXT := LPSU,LPTRK,FIN3;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % ASSN CREATES ASSIGNMENT BLOCKS AND SETS POINTERS TO LINK THINGS UP
    % AN ASSIGNMENT BLOCK IS TWO WORDS IN THE ARRAY "ASSNBLK".
    % ASSN IS ENTERED WITH NM CONTAINING THE NO. OF THE LOWEST LEVEL
    % BLOCK ASSOCIATED WITH THIS REQUEST AND THE CASE SWITCH WCH
    % INDICATING IF THE REQUEST IS PHASED,CONTIGUOUS,OR JUNK.
    % THE LEVEL BLK HAS POINTERS TO 3 LISTS OF ASSIGNMENT BLKS CORRES.
    % TO THE PHASED,CONTIGUOUS, AND JUNK ASSNBLKS ASSOCIATED WITH IT.
    % WITHIN EACH OF THESE 3 LISTS, THE ASSNBLKS ARE ORDERED BY NO. OF
    % SEGMENTS, WITH THE ASSIGNMENT REQUIRING THE GREATEST NO. AT THE
    % TOP.  FOUR CASES CAN OCCUR IN THIS PROCEDURE...
    %    1. THE LIST INTO WHICH THE NEW ASSIGNMENT BLOCK MUST GO IS NULL
    %                           ACTION
    %        THE NEW BLK IS CREATED AND THE CORRES. PTR IN THE
    %        LEVEL BLK IS POINTED TO IT.
    %    2. THE LIST IS NOT EMPTY AND THE NEW REQUEST MUST GO IN THE TOP
    %       OF THE LIST.
    %                           ACTION
    %        NEW BLK IS CREATED & THE LEVEL BLK PTR. POINTED TO IT.
    %         THE "NEXT BLK" PTR IN THE NEW BLK IS POINTED TO THE OLD
    %         "TOP OF THE LIST" BLK.
    %    3. THE LIST IS NOT EMPTY AND THE NEW REQUEST MUST BE INTERLEAVE
    %        IN IT BETWEEN ASSNBLKS A & B.
    %                           ACTION
    %          THE NEW BLK IS CREATED.
    %          "NEXT BLK" PTR. OF A POINTED TO NEW REQUEST.
    %          "NEXT BLK" PTR. OF NEW REQUEST POINTED TO B.
    %    4. THE LIST IS NOT EMPTY AND THE NEW REQUEST GOES AT THE END.
    %                           ACTION
    %        NEW BLK CREATED.
    %        "NXT BLK" PTR OF OLD "END OF LIST" ASSNBLK POINTED TO THE
    %        NEW REQUEST.
    % IN ALL CASES, THE ASSNBLK # OF THE PRECEEDING REQUEST FOR THE FILE
```

```
% IS ENTERED INTO THE NEW REQUEST.  IF THIS IS THE FIRST ASSNBLK FOR
% A FILE, THE ASSNBLK # IS ENTERED INTO THE FILE BLK FOR THAT FILE.
% WCH = CASE SWITCH.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PROCEDURE ASSN(WCH);
    INTEGER WCH;
    BEGIN
    INTEGER PP,QP,LASTONE;
    LABEL FILL1,FILL2,OTRO,FIN2;
    LASTONE:=0;
% LOOK AT THE PHASE, CONTIG, OR JUNK PTR OF THE LOWEST
% LEVEL BLK NECESSARY FOR THIS REQUEST.
CASE WCH OF
        BEGIN
        PP := PHASPTR(NN);
        PP := CONTIGPTR(NN);
        PP := JUNKPTR(NN);
       END;
   % IF THE PCJ PTR IN THE LEVEL BLOCK IS NULL CREATE THE NEW
   % ASSIGNMENT BLK AND SET THE PCJ PTR TO POINT TO IT.
   IF  PP  = 0 THEN GO TO FILL1 ;
 OTRO:
        %IF NO. OF SEGMENTS IN ASSNBLK POINTED TO IS > NEW GUY S
        %NO. OF SEGS, LOOK AT NXT ASSNBLK & COMPARE NO. OF SEGS.
        IF SEGNUM(PP) > SEGS THEN
        BEGIN
            %GET PTR TO NEXT ASSNBLK
            QP := NXTBL(PP);
            IF QP NEQ 0 THEN
               BEGIN
               LASTONE ← PP;
               PP ← QP;
               GO TO OTRO;
               END
              ELSE
              % CREATE NEW ASSNBLK AT END OF LIST OF OLD ASSNBLKS.
               BEGIN
               NXTBL(PP) ← NXASSN;     %POINT LASTONE ASSNBLK TO NEW.
  FILL2:       SEGNUM(NXASSN)←SEGS;    %ENTER NO. OF SEGS INTO NEW.
               PHNUM(NXASSN)←PHS;      %ENTER NO. OF PHASES.
               %IF THIS IS THE FIRST ASSNBLK FOR A FILE,
               %POINT THE FILE BLOCK TO IT.
               IF LASTIN=0 THEN FILASSN(NXFIL):=NXASSN ELSE
               NXTFRAG(LASTIN)←NXASSN;%POINT LAST CREATED TO NEW.
               LASTIN:=NXASSN;
               NXASSN ← NXASSN + 1;
               GO TO FIN2;
               END
       END
         ELSE %INTERLEAVE NEW ASSNBLK IN OLD LIST.
           BEGIN
           NXTBL(NXASSN) ← PP;
           % IF THIS IS THE FIRST ASSNBLK OF THIS TYPE FOR THIS
           % LEVEL BLK, POINT THE LEVEL BLK TO IT ELSE POINT THE
           % PRECEEDING ASSNBLK TO IT.
           IF LASTONE NEQ 0 THEN NXTBL(LASTONE):=NXASSN ELSE
               % POINT THE PHASE,CONTIG,OR JUNK PTR IN THE
               % LEVEL BLK TO THE ASSNBLK.
  FILL1:       CASE WCH OF
                    BEGIN
                    PHASPTR(NN) := NXASSN;
```

```
                    CONTIGPTR(NN):=NXASSN;
                    JUNKPTR(NN) := NXASSN;
                    END;
        , GO TO FILL2;
          END;
      FIN2:END ASSN;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FIND CREATES A NEW LEVEL BLK (BLK B) ACCORDING TO A REQUEST AND
% PTS THE ASSOCIATED BLK ON THE LEVEL ABOVE IT (BLK A) TO THE NEW
% LEVEL BLK.
% FIND IS ENTERED WITH NM CONTAINING THE LEVEL BLK NO. OF BLK A,
% AND WCH INDICATING WHICH PTR SHOULD BE TAKEN FROM BLK A.
% UPON EXIT FROM FIND, NN CONTAINS THE LEVEL BLOCK NO. OF BLK B.
% ASSOCIATED WITH EACH POINTER IN BLK A, IS A LIST OF LEVEL BLKS
% OF COMMON TYPE (E.G. FU,SU,TRK) IN THE ORDER IN WHICH THEY WERE
% CREATED, (LAST CREATED AT END OF LIST).
% THIS PROCEDURE LOOKS DOWN THE LIST AND 3 CASES CAN ARISE...
%     1. THE LIST IS EMPTY.
%                               ACTION
%           A NEW LEVEL BLK IS CREATED AND THE CORRES. PTR IN BLK A
%           IS SET TO POINT TO IT.
%     2. IT FINDS A LEVEL BLK IN THE LIST WITH THE SAME NO.
%        (I.E. FU,SU,TRK #).
%                               ACTION
%            NO NEW BLK IS CREATED.
%     3. IT FINDS NO LEVEL BLK IN THE LIST WITH THE SAME NO.
%                               ACTION
%            A NEW LEVEL BLK IS CREATED AND "NXT BLK" PTR OF THE OLD
%            "BOTTOM OF THE LIST" LEVEL BLK IS SET TO PT TO NEW BLK
% IN ALL CASES, NN IS ENTERED INTO AN ARRAY WHICH CONTAINS THE NO.S
% OF ALL LEVEL BLKS ASSOCIATED WITH THE PRESENT REQUEST.
% NUM = FU, SU, OR TRK NUMBER.
% TR = EXIT SWITCH.
% WCH = CASE SWITCH.
% VP = VIRTUAL PHYSICAL FLAG (1 IF PHYSICAL, 0 IF VIRTUAL).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
PROCEDURE FIND(LNXT,NUM,TR,WCH,VP);
    VALUE NUM;
SWITCH LNXT;
    INTEGER NUM,TR,WCH,VP;
    BEGIN
    INTEGER MM,NM;
    BOOLEAN FIRST1;
    LABEL NEWU,NXT;
    NM := NN;
% GET THE PTR TO THE NEXT LEVEL AS REQUIRED BY THE REQUEST.
    CASE WCH OF
       BEGIN
       NN := PHYPTR(NM);
       NN := NXTPTR(NM);
       NN := VSUPTR(NM);
       NN := VTRKPTR(NM);
       END;
    %IF THE PTR IS NULL CREATE A NEW BLOCK.
    IF  NN  = 0  THEN
       BEGIN
       FIRST1 := TRUE;
       GO TO NEWU;
       END ;
    NXT:
       %IF FU,SU,TRK# AGREE, YOU VE GOT NN...EXIT FIND.
```

```
            IF NUMBPTR(NN) # NUM  THEN
            BEGIN
               %IF PTR TO NEXT BLOCK NOT 0, GET NEXT BLOCK & GO TO NXT
                MM := NXTPTR(NN);
                IF MM  # 0 THEN
                     BEGIN
                     NN ← MM;
                     GO TO NXT;
                     END
                  ELSE  %NO EXISTING BLKS AGREED, CREATE NEW ONE.
                     BEGIN
                     FIRST1 := FALSE;
                     NXTPTR(NN) ← NXTBLK; %POINT LAST BLOCK TO NEW ONE.
        NEWU:        NUMBPTR(NXTBLK) ← NUM;%ENTER FD,SU,OR TRK NUMBER.
               VPPTR(NXTBLK) := VP;  %ENTER VIRTUAL PHYSICAL BIT.
                     NN ← NXTBLK;              %POINT NN AT THIS NEW BLOCK.
                     NXTBLK ← NXTBLK +1;
                     % IF THIS IS THE FIRST LEVEL BLK OF THIS TYPE
                     % FOR THE ABOVE LEVEL BLK, PT THE ABOVE TO THIS NEW
                     IF FIRST1 THEN
                     CASE WCH OF
                          BEGIN
                          PHYPTR(NM) := NN;
                          NXTPTR(NM) := NN;
                          VSUPTR(NM) := NN;
                          VTRKPTR(NM) := NN;
                          END;
                     END;
            END;
            % ENTER NN IN THE ARRAY THAT PTS TO ALL BLKS
            % CONNECTED WITH THIS REQUEST.
            ADDSEGS[AS] := NN;
            AS := AS + 1;
            GO TO LNXT[TP+1];                %BRANCH ACCORDING TO SWITCH.
         END FIND;
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %%%%%%%% BEGIN MAIN BODY OF PROGRAM %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      FOR M:=0 STEP 1 UNTIL 63 DO
        FOR N:=0 STEP 1 UNTIL 191 DO
         BEGIN
          ASSNBLK[M,N]:=0;
          PTRBLK[M,N]:=0;
          TOPBLK[M,N]:=0;
          FILBLK[M,N]:=0;
         END;
      % INITIALIZE ARRAY ROW POINTERS.
      NXTBLK ← 2;
      NXASSN ← 1;
      NXFIL  ← 1;
      NEXT := 1;
                  %SET RECORD PTR TO RECORD #1 OF PASSFILE
      TOP := TRUE;
RD:   % READ THE NEXT 30 WORDS FROM THE FILE NAME TABLE BLOCK.
      READ(PASSFILE[NEXT],30,HOLDIT[*]);
      % ENTER PHYSICAL RECORD NO. OF THE NEXT BLK FOR NEXT READ.
      NEXT ← SYSLINK;
      % IF THIS IS A "NEW FILE" READ (I.E. IF THERE IS NO
      % FILE HEADER IN THIS PHYSICAL RECORD) BEGIN READING 2 WORD REQUESTS
      IF NOT TOP THEN
      BEGIN
```

```
    I := 1;
    GO TO LP9;
    END;
    I := 9;
    % READ FILE NAME INTO NEXT FILE BLOCK.
    REPLACE(AFILNAME(NXFIL),0,HOLDIT[1],0,1);
    REPLACE(BFILNAME(NXFIL),0,HOLDIT[2],0,1);
    REPLACE(CFILNAME(NXFIL),0,HOLDIT[3],0,1);
    % READ IN NO. OF SEGS/RECD AND TOTAL NO. OF SEGS FOR THIS FILE.
    SEGSRECD(NXFIL) ← ASEGSRECD;
    TOTSEGS(NXFIL) ← ATOTSEGS;
    % ENTER POST AND PRE PROCESS LOCATIONS INTO FILE BLK.
    PREPTR(NXFIL) ← APREPTR;
    POSTPTR(NXFIL) ← APOSTPTR;
    % IF YOU HAVE PROCESSED ALL 29 REQUESTS IN THIS PHYSICAL RECORD
    % GO GET ANOTHER ONE.
LP9:IF  I = 29 THEN
        BEGIN
        TOP := FALSE;
        GO TO RD;
        END;
    % INITIALIZE NN TO POINT TO TOP BLK.
    NN := 1;
    % INITIALIZE ADDSEGS ARRAY INDEX TO 0.
    AS := 0;
    A:=HOLDIT[1];
    B:=HOLDIT[1+1];
    % IF THIS IS THE END OF THIS FILE, SEE IF THERE ARE OTHER FILES.
    IF A = 0 AND B = 0 THEN
        % IF THERE IS ANOTHERE FILE PREPARE TO FORM FILE BLOCK.
        IF NEXT = 0 THEN GO TO FIN4 ELSE
           BEGIN
           TOP := TRUE;              % INDICATE NEW FILE READ.
           NXFIL ← NXFIL + 1;        % POINT TO NEXT FILE BLK.
           LASTIN := 0;              % INITIALIZE LASTIN.
           GO TO RD;
           END;
    %IF A PHYSICAL FU IS NOT REQUIRED GO TO LVFU.
    IF PFU = 0 THEN GO TO LVFU;
    FIND(LNXT,FUNUM,0,0,1);
LPSU:
    % IF A PHYSICAL SU NOT REQUIRED GO TO LVSU.
    IF PSU = 0 THEN GO TO LVSU;
    FIND(LNXT,SUNUM,1,0,1);
LPTRK:
    % IF A PHYSICAL TRACK IS NOT REQUIRED, GO TO LVTRK.
    IF PTRK =0 THEN GO TO LVTRK;
    FIND(LNXT,TRKNUM,2,0,1);
LVFU:
    % IF A VIRTUAL FU NOT REQUIRED GO TO LVSU.
    IF VFU = 0 THEN GO TO LVSU;
    FIND(LNXT,FUNUM,0,1,0);
LVSU:
    % IF A VIRTUAL SU IS NOT REQUIRED GO TO LVTRK.
    IF VSU = 0 THEN  GO TO LVTRK;
    FIND(LNXT,SUNUM,1,2,0);
LVTRK:
    % IF A VIRTUAL TRACK IS NOT REQUIRED GO TO FINL.
    IF VTRK = 0 THEN GO TO FIN3;
    FIND(LNXT,TRKNUM,2,3,0);
FIN3:
```

```
    % IF THE NEW REQUEST IS PHASED SET THE CASE SWITCH TO 0 FOR ASSN.
    IF PH = 1 THEN ASSN(0) ELSE
    %IF REQUEST IS CONTIGUOUS SET THE CASE SWITCH IN ASSN TO 1        .
    IF CONT = 1 THEN ASSN(1) ELSE ASSN(2);
    % ADD TOP BLK TO ADDSEGS ARRAY.
    ADDSEGS[AS] := 1;
    % ADD NO. OF SEGS TO TOTAL IN ALL LEVEL BLKS THAT LEAD
    % TO THE NEW ASSNBLK.
    FOR AS := AS STEP -1 UNTIL 0 DO
        BEGIN
        NUMSEGS(ADDSEGS[AS]) := NUMSEGS(ADDSEGS[AS]) + SEGS;
        NUMFRAGS(ADDSEGS[AS]) := NUMFRAGS(ADDSEGS[AS]) +  1;
        END;
    I := I+2;    % POINT I TO NEXT 2 WORD REQUEST IN HOLD ARRAY.
    GO TO LP9;
FIN4:
END TREEBUILD;
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PROCEDURE ALLOCATE USES THE ALLOCATION TREE AND ITS OWN FREE SPACE
% ARRAYS TO ALLOCATE SPACE ON THE DISK FOR FILE REQUESTS.
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  PROCEDURE ALLOCATE;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
BEGIN
  PROCEDURE ASSIGN (ASSNBLKPTR,WHICHMAP,PHASE,THISMAP);
    INTEGER ASSNBLKPTR,WHICHMAP,PHASE,THISMAP;
    BEGIN
      INTEGER MAPADDR,NEWMAP;
      LABEL FIN;
      ASSNBLKEU(ASSNBLKPTR):=MEU(WHICHMAP);
      ASSNBLKSU(ASSNBLKPTR):=MSU(WHICHMAP);
      ASSNBLKTRK(ASSNBLKPTR):=ENTIER(PHASE/NUMSEGSPERTRK);
      ASSNBLKSEG(ASSNBLKPTR):=PHASE MOD(NUMSEGSPERTRK);
      NUMSEGINMAP(WHICHMAP):=NUMSEGINMAP(WHICHMAP)-
       SEGNUM(ASSNBLKPTR);
      IF FIRSTSEG(THISMAP)=PHASE THEN
          IF NUMSEG(THISMAP) NEQ SEGNUM(ASSNBLKPTR) THEN
            BEGIN
              FIRSTSEG(THISMAP):=PHASE+SEGNUM(ASSNBLKPTR);
              GO TO FIN;
            END
          ELSE
            BEGIN
              IF THISMAP NEQ MAPPTR(WHICHMAP) THEN
                BEGIN
                  MAPADDR:=MAPPTR(WHICHMAP);
                  WHILE NEXTMAPBLOCK(MAPADDR) NEQ THISMAP DO
                    BEGIN
                      MAPADDR:=NEXTMAPBLOCK(MAPADDR);
                      NEXTMAPBLOCK(MAPADDR):=NEXTMAPBLOCK(THISMAP);
                    END;
                END
              ELSE
```

```
                MAPPTR(WHICHMAP):=NEXTMAPBLOCK(THISMAP);
              FREEMAPSPACE (THISMAP);
              NMAPFRAGMENTS(WHICHMAP):=NMAPFRAGMENTS(WHICHMAP)-1;
              GO TO FIN;
            END
      ELSE
        BEGIN
          IF PHASE+SEGNUM(ASSNBLKPTR) NEQ FIRSTSEG(THISMAP)+NUMSEG
           (THISMAP) THEN
            BEGIN
             NEWMAP:=NEXTMAPBLOCK(THISMAP);
              GETMAPSPACE (NEWMAP);
              FIRSTSEG(NEWMAP):=PHASE+SEGNUM(ASSNBLKPTR);
              NUMSEG(NEWMAP):=FIRSTSEG(THISMAP)+NUMSEG(THISMAP)-
               FIRSTSEG(NEWMAP);
              NEXTMAPBLOCK(NEWMAP):=NEXTMAPBLOCK(THISMAP);
              NEXTMAPBLOCK(THISMAP):=NEWMAP;
             NMAPFRAGMENTS(WHICHMAP):=NMAPFRAGMENTS(WHICHMAP)+1;
            END;
          NUMSEG(THISMAP):=PHASE-FIRSTSEG(THISMAP);
        END;
FIN:
    END ASSIGN;
%
%
%
  INTEGER PROCEDURE RELEASER(ASSNBLKPTR,WHICHMAP);
    INTEGER ASSNBLKPTR,WHICHMAP;
    BEGIN
      INTEGER PHASE,TOTALSEG,NEXTMAP,MAPADDR,NEWMAP;
      LABEL FIN,LOOP;
      NUMSEGINMAP(WHICHMAP):=NUMSEGINMAP(WHICHMAP)+SEGNUM(ASSNBLKPTR);
      PHASE:=ASSNBLKTRK(ASSNBLKPTR)*NUMSEGSPERTRK+ASSNBLKSEG
       (ASSNBLKPTR);
      TOTALSEG:=SEGNUM(ASSNBLKPTR);
      NEXTMAP:=MAPADDR:=MAPPTR(WHICHMAP);
LOOP:
      IF NEXTMAP=0 OR PHASE+TOTALSEG LSS FIRSTSEG(NEXTMAP) THEN
        BEGIN
          GETMAPSPACE(NEWMAP);
          FIRSTSEG(NEWMAP):=PHASE;
          NUMSEG(NEWMAP):=TOTALSEG;
          NEXTMAPBLOCK(NEWMAP):=NEXTMAP;
          NMAPFRAGMENTS(WHICHMAP):=NMAPFRAGMENTS(WHICHMAP)+1;
          IF MAPADDR=0 THEN
            DO
              BEGIN
                MAPADDR:=NEXTMAPBLOCK(MAPADDR);
                NEXTMAPBLOCK(MAPADDR):=NEWMAP;
              END
            UNTIL NEXTMAPBLOCK(MAPADDR)=NEXTMAP
          ELSE
              MAPPTR(WHICHMAP):=NEWMAP;
          RELEASER:=NEWMAP;
          GO TO FIN;
        END
      ELSE
        BEGIN
          IF PHASE+TOTALSEG NEQ FIRSTSEG(NEXTMAP) THEN
              IF PHASE=FIRSTSEG(NEXTMAP)+NUMSEG(NEXTMAP) THEN
                  IF NEXTMAPBLOCK(NEXTMAP) NEQ 0 THEN
```

```
                    BEGIN
                      IF FIRSTSEG(NEXTMAP)+NUMSEG(NEXTMAP)+TOTALSEG
                       =FIRSTSEG(NEXTMAPBLOCK(NEXTMAP)) THEN
                        BEGIN
                          NUMSEG(NEXTMAP):=NUMSEG(NEXTMAP)+TOTALSEG+
                           NUMSEG(NEXTMAPBLOCK(NEXTMAP));
                          NEXTMAPBLOCK(NEXTMAP):=NEXTMAPBLOCK
                           (NEXTMAPBLOCK(NEXTMAP));
                          FREEMAPSPACE(NEXTMAPBLOCK(NEXTMAP));
                          NMAPFRAGMENTS(WHICHMAP):=
                           NMAPFRAGMENTS(WHICHMAP)-1;
                        END;
                    END
                ELSE
                  BEGIN
                    NEXTMAP:=NEXTMAPBLOCK(NEXTMAP);
                    GO TO LOOP;
                  END
            ELSE
                FIRSTSEG(NEXTMAP):=PHASE;
            NUMSEG(NEXTMAP):=NUMSEG(NEXTMAP)+TOTALSEG;
            RELEASER:=NEXTMAP;
          END;
FIN;
    END RELEASER;
%
%
%
  PROCEDURE ALLOCJUNK (ASSNBLKPTR,WHICHMAP);
    INTEGER ASSNBLKPTR,WHICHMAP;
    BEGIN
      INTEGER NEWASSNBLK,NBLOCK;
      IF ASSNBLKPTR NEQ 0 THEN
        BEGIN
          NBLOCK←MAPPTR(WHICHMAP);
          IF NUMSEG(NBLOCK)<SEGNUM(ASSNBLKPTR) THEN
            BEGIN
              NEWASSNBLK:=NXTBL(ASSNBLKPTR);
              GETASSNBLKSPACE (NEWASSNBLK);
              NXTBL(NEWASSNBLK):=NXTBL(ASSNBLKPTR);
              NXTFRAG(NEWASSNBLK):=NXTFRAG(ASSNBLKPTR);
              NXTBL(ASSNBLKPTR):=NEWASSNBLK;
                        NXTFRAG(ASSNBLKPTR):=NEWASSNBLK;
              SEGNUM(NEWASSNBLK)←SEGNUM(ASSNBLKPTR)-NUMSEG(NBLOCK);
              SEGNUM(ASSNBLKPTR)←NUMSEG(NBLOCK);
            END;
          ASSIGN (ASSNBLKPTR,WHICHMAP,FIRSTSEG(NBLOCK),NBLOCK);
          ALLOCJUNK (NXTBL(ASSNBLKPTR),WHICHMAP);
        END;
    END ALLOCJUNK;
%
%
%
  BOOLEAN PROCEDURE ALLOCCONTIG(ASSNBLKPTR,WHICHMAP);
    INTEGER ASSNBLKPTR,WHICHMAP;
    BEGIN
      INTEGER NBLOCK;
      LABEL LOOP,FIN;
      IF ASSNBLKPTR≠0 THEN
        BEGIN
          NBLOCK←MAPPTR(WHICHMAP);
```

```
LOOP;
          IF NBLOCK≠0 THEN
              IF NUMSEG(NBLOCK)≥SEGNUM(ASSNBLKPTR) THEN
                BEGIN
                  ASSIGN(ASSNBLKPTR,WHICHMAP,FIRSTSEG(NBLOCK),NBLOCK);
                  IF ALLOCCONTIG(NXTBL(ASSNBLKPTR),WHICHMAP)
                   THEN
                    BEGIN
                      ALLOCCONTIG:=TRUE;
                      GO TO FIN;
                    END
                  ELSE
                    BEGIN
                      NBLOCK:=NEXTMAPBLOCK(RELEASE(ASSNBLKPTR,
                       WHICHMAP));
                      GO TO LOOP;
                    END;
                END
              ELSE
                BEGIN
                  NBLOCK←NEXTMAPBLOCK(NBLOCK);
                  GO TO LOOP;
                END
            ELSE
              BEGIN
                ALLOCCONTIG←FALSE;
                GO TO FIN;
              END;
        END
      ELSE
          ALLOCCONTIG←TRUE;
FIN;
    END ALLOCCONTIG;
%
%
%
  BOOLEAN PROCEDURE PALLOC (PHASE1,ASSNBLKPTR,WHICHMAP);
    VALUE PHASE1,ASSNBLKPTR,WHICHMAP;
    INTEGER PHASE1,ASSNBLKPTR,WHICHMAP;
    BEGIN
      INTEGER NEXTMAP,PHASE;
      LABEL LOOP1,LOOP2,FINTRUE,FINFALSE,FIN;
LOOP1:
      IF ASSNBLKPTR≠0 THEN
        BEGIN
          NEXTMAP←MAPPTR(WHICHMAP);
          PHASE:=PHASE1+PHNUM(ASSNBLKPTR) MOD NUMSEGSPERTRK;
          IF NEXTMAP≠0 THEN
            BEGIN
              IF FIRSTSEG(NEXTMAP)≤PHASE THEN
                  IF FIRSTSEG(NEXTMAP)+NUMSEG(NEXTMAP)<PHASE THEN
                    BEGIN
LOOP2:
                      NEXTMAP←NEXTMAPBLOCK(NEXTMAP);
                      GO TO LOOP1;
                    END
                  ELSE
                    BEGIN
                      IF FIRSTSEG(NEXTMAP)+NUMSEG(NEXTMAP)≥PHASE+SEGNUM
                       (ASSNBLKPTR) THEN
                        BEGIN
```

```
                        ASSIGN(ASSNBLKPTR,WHICHMAP,PHASE,NEXTMAP);
                        IF PALLOC(PHASE1,NXTBL(ASSNBLKPTR),
                         WHICHMAP) THEN
                            GO TO FINTRUE
                        ELSE
                    NEXTMAP:=RELEASER(ASSNBLKPTR,WHICHMAP);
                        END;
                      IF PHASE GEQ NUMSEGINMAP(WHICHMAP) THEN
                        GO TO FINFALSE
                      ELSE
                        GO TO LOOP1;
                    END
                ELSE
                    BEGIN
                      PHASE:=PHASE+NUMSEGSPERTRK;
                      IF PHASE=NUMSEGINMAP(WHICHMAP) THEN
                        GO TO FINFALSE
                      ELSE
                        GO TO LOOP2;
                    END;
            END
          ELSE
            GO TO FINFALSE;
        END;
FINTRUE: PALLOC:=TRUE;
         GO TO FIN;
FINFALSE: PALLOC:=FALSE;
FIN:
    END PALLOC;
%
%
%
  BOOLEAN PROCEDURE ALLOCPHASED(ASSNBLKPTR,WHICHMAP);
    INTEGER ASSNBLKPTR,WHICHMAP;
    BEGIN
      INTEGER NBLOCK,PHASE;
      LABEL LOOP,FIN;
      NBLOCK:=MAPPTR(WHICHMAP);
LOOP:
      PHASE:=FIRSTSEG(NBLOCK);
      IF NOT PALLOC(PHASE,ASSNBLKPTR,WHICHMAP) THEN
        BEGIN
          NBLOCK:=NEXTMAPBLOCK(NBLOCK);
          IF NBLOCK#0 THEN
              GO TO LOOP
          ELSE
            BEGIN
              ALLOCPHASED:=FALSE;
              GO TO FIN;
            END;
        END
      ELSE
          ALLOCPHASED:=TRUE;
FIN:
    END ALLOCPHASED;
%
%
%
  BOOLEAN PROCEDURE ALLOCABSPHASED (ASSNBLKPTR,WHICHMAP);
    INTEGER ASSNBLKPTR,WHICHMAP;
    BEGIN
```

```
      INTEGER NBLOCK,PHASE,SIZE;
      LABEL LOOP,FIN,FINTRUE;
      IF ASSNBLKPTR#0 THEN
        BEGIN
          NBLOCK←MAPPTR(WHICHMAP);
          PHASE:=PHNUM(ASSNBLKPTR);
          SIZE←SEGNUM(ASSNBLKPTR);
LOOP:
          IF NBLOCK#0 THEN
              IF FIRSTSEG(NBLOCK)≤PHASE THEN
                  IF FIRSTSEG(NBLOCK)+NUMSEG(NBLOCK)≥PHASE+SIZE THEN
                    BEGIN
                      ASSIGN(ASSNBLKPTR,WHICHMAP,PHASE,NBLOCK,
             IF ALLOCABSPHASED(NXTBL(ASSNBLKPTR),
                       WHICHMAP) THEN
                          GO TO FINTRUE;
%                     ELSE
%                         RELEASER(ASSNBLKPTR,WHICHMAP);
                    END
                  ELSE
                    BEGIN
                      NBLOCK←NEXTMAPBLOCK(NBLOCK);
                      GO TO LOOP;
                    END;
          ALLOCABSPHASED←FALSE;
        END;
FINTRUE:
      ALLOCABSPHASED:=TRUE;
    END ALLOCABSPHASED;
%
%
%
  PROCEDURE DEALLOCATELIST(PTR);
    INTEGER PTR;
    IF PTR NEQ 0 THEN
      BEGIN
%       RELEASER(PTR,WHICHMAPNUM(ASSNBLKSU(PTR)),
%        ASSNBLKSU(PTR));
        DEALLOCATELIST(NXTBL(PTR));
      END DEALLOCATELIST;
%
%
%
  PROCEDURE DEALLOCATETRK(PTR);
    INTEGER PTR;
      IF PTR NEQ 0 THEN
        BEGIN
          DEALLOCATELIST(PHASEDPTR(PTR));
          DEALLOCATELIST(CONTGPTR(PTR));
          DEALLOCATELIST(JNKPTR(PTR));
          DEALLOCATETRK(NXTPTR(PTR));
        END DEALLOCATETRK;
%
%
%
  PROCEDURE DEALLOCATESU(PTR);
    INTEGER PTR;
      IF PTR NEQ 0 THEN
        BEGIN
          DEALLOCATETRK(PHYSICALTRK(PTR));
          DEALLOCATETRK(VIRTUALTRK(PTR));
```

```
          DEALLOCATELIST(PHASEDPTR(PTR));
          DEALLOCATELIST(CONTGPTR(PTR));
          DEALLOCATELIST(JNKPTR(PTR));
          DEALLOCATESU(NXTPTR(PTR));
        END DEALLOCATESU;
%
%
%
  PROCEDURE DEALLOCATEEU(PTR);
    INTEGER PTR;
    IF PTR NEQ 0 THEN
      BEGIN
        DEALLOCATESU(PHYSICALSU(PTR));
        DEALLOCATESU(VIRTUALSU(PTR));
        DEALLOCATETRK(VIRTUALTRK(PTR));
        DEALLOCATELIST(PHASEDPTR(PTR));
        DEALLOCATELIST(CONTGPTR(PTR));
        DEALLOCATELIST(JNKPTR(PTR));
        DEALLOCATEEU(NXTPTR(PTR));
      END DEALLOCATEEU;
%
%
%
  PROCEDURE DEALLOCATEDISK;
    BEGIN
      DEALLOCATEEU(PHYSICALEU(TOPBLOCK));
      DEALLOCATEEU(VIRTUALEU(TOPBLOCK));
      DEALLOCATESU(VIRTUALSU(TOPBLOCK));
      DEALLOCATETRK(VIRTUALTRK(TOPBLOCK));
      DEALLOCATELIST(PHASEDPTR(TOPBLOCK));
      DEALLOCATELIST(CONTGPTR(TOPBLOCK));
      DEALLOCATELIST(JNKPTR(TOPBLOCK));
    END DEALLOCATEDISK;
%
%
%
  PROCEDURE FUSEJUNKALLOC(ASSNBLKPTR,FIRSTEU,LASTEU,FIRSTSU,LASTSU);
    INTEGER ASSNBLKPTR,FIRSTEU,LASTEU,FIRSTSU,LASTSU;
    BEGIN
      INTEGER NBLOCK,WHICHMAP,EU,SU;
      LABEL FIN;
      INTEGER NEWASSNBLK;
      IF ASSNBLKPTR≠0 THEN
        BEGIN
          FOR EU←FIRSTEU STEP (IF FIRSTEU≤LASTEU THEN 1 ELSE -1) UNTIL
           LASTEU DO
              BEGIN
                WHICHMAP:=MAPBLKPTRARRAY[EU,SU];
                NBLOCK←MAPPTR(WHICHMAP);
                IF NUMSEGINMAP(WHICHMAP)≠0 THEN
                  BEGIN
                    IF NUMSEG(NBLOCK)<SEGNUM(ASSNBLKPTR) THEN
                      BEGIN
                        GETASSNBLKSPACE(NEWASSNBLK);
                        NXTBL(NEWASSNBLK):=NXTBL(ASSNBLKPTR);
                 NXTFRAG(NEWASSNBLK):=NEWASSNBLK;
                 NXTBL(ASSNBLKPTR):=NEWASSNBLK;
                        SEGNUM(NEWASSNBLK)←SEGNUM(ASSNBLKPTR)+
                         NUMSEG(NBLOCK);
                        SEGNUM(ASSNBLKPTR)←NUMSEG(NBLOCK);
                      END;
```

```
                    ASSIGN(ASSNBLKPTR,WHICHMAP,FIRSTSEG(NBLOCK),NBLOCK);
                    NUMSEGINMAP(WHICHMAP)←NUMSEGINMAP(WHICHMAP)-SEGNUM
                     (ASSNBLKPTR);
                    GO TO FIN;
                  END;
              END;
%         ERROR TERMINATION
        END;
FIN:
    END EUSUJUNKALLOC;
%
%
%
  BOOLEAN PROCEDURE EUSUCONTIGALLOC(ASSNBLKPTR,FIRSTEU,LASTEU,FIRSTSU,
  LASTSU);
    INTEGER ASSNBLKPTR,FIRSTEU,LASTEU,FIRSTSU,LASTSU;
    BEGIN
      INTEGER NBLOCK,WHICHMAP,EU,SU;
      LABEL FINTRUE,LOOP,FIN;
      IF ASSNBLKPTR=0 THEN
          EUSUCONTIGALLOC←TRUE
      ELSE
          FOR EU←FIRSTEU STEP (IF FIRSTEU LEQ LASTEU THEN 1
               ELSE -1) UNTIL LASTEU DO
            FOR SU←FIRSTSU STEP 1 UNTIL LASTSU DO
              BEGIN
                WHICHMAP:=MAPBLKPTRARRAY[EU,SU];
                NBLOCK←MAPPTR(WHICHMAP);
                IF NUMSEGINMAP(WHICHMAP)≥SEGNUM(ASSNBLKPTR) THEN
LOOP:
                    IF NBLOCK≠0 THEN
                      IF NUMSEG(NBLOCK)≥SEGNUM(ASSNBLKPTR) THEN
                        BEGIN
                          ASSIGN(ASSNBLKPTR,WHICHMAP,FIRSTSEG(NBLOCK),
                           NBLOCK);
                          NUMSEGINMAP(WHICHMAP)←NUMSEGINMAP(WHICHMAP)-
                           SEGNUM(ASSNBLKPTR);
                    IF EUSUCONTIGALLOC(NXTEL(ASSNBLKPTR
                     ),LASTEU,FIRSTEU,FIRSTEU,LASTEU) THEN
                        GO TO FINTRUE;
                      NBLOCK:=NEXTMAPBLOCK(RELEASER(ASSNBLKPTR,
                       WHICHMAP));
                      NUMSEGINMAP(WHICHMAP)←NUMSEGINMAP(WHICHMAP)+
                       SEGNUM(ASSNBLKPTR);
                      GO TO LOOP;
                    END
                  ELSE
                    BEGIN
                      NBLOCK←NEXTMAPBLOCK(NBLOCK);
                      GO TO LOOP;
                    END;
              END;
        EUSUCONTIGALLOC←FALSE;
       GO TO FIN;
FINTRUE: EUSUCONTIGALLOC←TRUE;
FIN:
  END EUSUCONTIGALLOC;
%
%
%
  BOOLEAN PROCEDURE ALLOCATEPHASED(PHASEDPTR,BLOCKPTR);
```

```
    INTEGER PHASEDPTR,BLOCKPTR;
    BEGIN
    END;
%
%
%
  BOOLEAN PROCEDURE ALLOCATECONTRK(FIRSTEU,LASTEU,FIRSTSU,LASTSU,
   BLOCKPTR);
    INTEGER FIRSTEU,LASTEU,FIRSTSU,LASTSU,BLOCKPTR;
    BEGIN
    FORMAT FALLOCATECONTRK ("ERROR IN TRACK ALLOCATION.  ALLOCATION ",
                "TERMINATED.");
        IF BLOCKPTR=0 THEN
            ALLOCATECONTRK:=TRUE
        ELSE
    END ALLOCATECONTRK;
%
%
%
  BOOLEAN PROCEDURE ALLOCATECONSUC(FIRSTEU,LASTEU,BLOCKPTR);
    INTEGER FIRSTEU,LASTEU,BLOCKPTR;
    BEGIN
      INTEGER WHICHMAP,SU,EU,FIRSTSU,LASTSU;
      LABEL FIN;
      IF BLOCKPTR NEQ 0 THEN
        BEGIN
          IF VPPTR(BLOCKPTR)=0 THEN
              FIRSTEU:=LASTEU:=NUMBPTR(BLOCKPTR)
          ELSE
            BEGIN
              FIRSTSU:=FIRSTSUNUM;
              LASTSU:=LASTSUNUM;
            END;
          FOR EU:=FIRSTEU STEP 1 UNTIL LASTEU DO
            FOR SU:=FIRSTSU STEP 1 UNTIL LASTSU DO
              BEGIN
                WHICHMAP:=MAPBLKPTRARRAY[EU,SU];
                IF NOT (MLOCK(WHICHMAP)=0 OR NUMSEGS(BLOCKPTR) LSS
                 NUMSEGINMAP(WHICHMAP)) THEN
                  BEGIN
                    IF ALLOCATECONTRK(EU,EU,SU,SU,PHYSICALTRK(BLOCKPTR))
                     THEN
                      BEGIN
                        IF ALLOCATEPHASED(PHASEDPTR(BLOCKPTR),
                         WHICHMAP) THEN
                          BEGIN
                            IF ALLOCCONTIG(CONTIGPTR(BLOCKPTR),
                             WHICHMAP) THEN
                              BEGIN
                                ALLOCJUNK(JUNKPTR(BLOCKPTR),
                                 WHICHMAP);
                                IF VPPTR(BLOCKPTR)=0 THEN
                                  BEGIN
                                    MLOCK(WHICHMAP):=1;
                                    NUMAVAILSUSEGS(EU,SU):=
                                     NUMAVAILSUSEGS(EU,SU)-NUMSEGS
                                     (BLOCKPTR);
                                  END;
                                IF ALLOCATECONSUC(FIRSTEU,LASTEU,
                                 NXTPTR(BLOCKPTR)) THEN
                                  BEGIN
```

```
                                    ALLOCATEONSU:=TRUE;
                                    GO TO FIN;
                                  END
                                ELSE
                                  BEGIN
                                    DEALLOCATELIST(CONTIGPTR
                                     (BLOCKPTR));
                                    DEALLOCATELIST(JUNKPTR
                                     (BLOCKPTR));
                                    MLOCK(WHICHMAP):=0;
                                    NUMAVAILSUSEGS(EU,SU):=
                                     NUMAVAILSUSEGS(EU,SU)+
                                     NUMSEGS(BLOCKPTR);
                                  END;
                              END;
                            DEALLOCATELIST(PHASEDPTR(BLOCKPTR));
                          END;
                        DEALLOCATETRK(VIRTUALTRK(BLOCKPTR));
                      END;
                    DEALLOCATETRK(PHYSICALTRK(BLOCKPTR));
                  END;
              END;
          ALLOCATEONSU:=FALSE;
          GO TO FIN;
        END;
        ALLOCATEONSU:=TRUE;
FIN;
   END ALLOCATEONSU;
%
%
%
  BOOLEAN PROCEDURE ALLOCATEONEU(BLOCKPTR);
    INTEGER BLOCKPTR;
    BEGIN
      INTEGER EU,SU,FIRSTEU,LASTEU,WHICHMAP;
      LABEL FIN,FINTRUE;
      IF BLOCKPTR NEQ 0 THEN
        BEGIN
          IF VPPTR(BLOCKPTR) = 0 THEN
              FIRSTEU:=LASTEU:=NUMAPTR(BLOCKPTR)
          ELSE
            BEGIN
              FIRSTEU:=FIRSTEUNUM;
              LASTEU:=LASTEUNUM;
            END;
          FOR EU:=FIRSTEU STEP 1 UNTIL LASTEU DO
            BEGIN
              WHICHMAP:=WHICHMAPNUM(EU,SU);
              IF NOT (MLOCK(WHICHMAP)=1 OR NUMSEGS(BLOCKPTR) LSS
               NUMAVAILSUSEGS(EU,SU)) THEN
                  IF ALLOCATEONSU(EU,SU,PHYSICALSU(BLOCKPTR)) THEN
                       IF ALLOCATEONSU(EU,SU,VIRTUALSU(BLOCKPTR)) THEN
                         BEGIN
                           IF ALLOCATEONTRK(EU,SU,FIRSTSUNUM,LASTSUNUM,
                            VIRTUALTRK(BLOCKPTR)) THEN
                             BEGIN
                               FOR SU:=FIRSTSUNUM STEP 1 UNTIL LASTSUNUM
                                DO
                                   IF ALLOCATEPHASED(PHASEDPTR(BLOCKPTR)
                                    ,WHICHMAP) THEN
                                     IF EUSUCONTIGALLOC(CONTIGPTR
```

```
                                        (BLOCKPTR),FU,SU,FIRSTSUNUM,
                                        LASTSUNUM) THEN
                                         BEGIN
                                           FUSUJUNKALLOC(JUNKPTR(BLOCKPTR)
                                            ,FU,SU,FIRSTSUNUM,LASTSUNUM);
                                           IF VPPTR(BLOCKPTR)=1 THEN
                                             BEGIN
                                               MLOCK(WHICHMAP):=1;
                                               NUMAVAILSUSEGS(FU,SU):=
                                                NUMAVAILSUSEGS(FU,SU)-
                                        NUMSEGS(BLOCKPTR);
                                             END;
                                           IF ALLOCATEONFU(NXTPTP(BLOCK TR)
                                            ) THEN
                                             GO TO FINTRUE
                                           ELSE
                                             BEGIN
                                               DEALLOCATELIST(JUNKPTR
                                                (BLOCKPTR));
                                               DEALLOCATELIST(CONFIGPTR
                                                (BLOCKPTR));
                                               MLOCK(WHICHMAP):=0;
                                               NUMAVAILSUSEGS(FU,SU):=
                                                NUMAVAILSUSEGS(FU,SU)+
                                                NUMSEGS(BLOCKPTR);
                                             END;
                                         END;
                                    DEALLOCATELIST(PHASEDPTR(BLOCKPTR));
                                  DEALLOCATETRK(VIRTUALTRK(BLOCKPTR));
                                END;
                              DEALLOCATESU(VIRTUALSU(BLOCKPTR));
                            END;
                         DEALLOCATESU(PHYSICALSU(BLOCKPTR));
                    END;
                  ALLOCATEONFU:=FALSE;
                  GO TO FIN;
                END;
FINTRUE:
       ALLOCATEONFU:=TRUE;
FIN:
    END ALLOCATEONFU;
%
%
%
  BOOLEAN PROCEDURE ALLOCATEONDISK;
    BEGIN
      INTEGER FU,SU;
      LABEL FIN,JUMP;
      IF NUMSEGS(TOPBLOCK) LEQ AMOUNTOFFREEDISK THEN
           IF ALLOCATEONFU(PHYSICALFU(TOPBLOCK)) THEN
                IF ALLOCATEONFU(VIRTUALFU(TOPBLOCK)) THEN
                  BEGIN
                    IF ALLOCATEONSU(FIRSTFUNUM,LASTFUNUM,VIRTUALSU
                     (TOPBLOCK)) THEN
                      BEGIN
                        IF ALLOCATEONTRK(FIRSTFUNUM,LASTFUNUM,FIRSTSUNUM,
                         LASTSUNUM,VIRTUALTRK(TOPBLOCK)) THEN
                          BEGIN
                            FOR FU:=FIRSTFUNUM STEP 1 UNTIL LASTFUNUM DO
                      FOR SU:=FIRSTSUNUM STEP 1 UNTIL LASTSUNUM
                       DO
```

```
                                IF ALLOCATEPHASED(PHASEDPTR(TOPBLOCK),
                                WHICHMAPNUM(EU,SU)) THEN
                                  BEGIN
                                    IF EUSUCONTIGALLOC(CONTIGPTR
                                     (TOPBLOCK),FIRSTEUNUM,LASTEUNUM,
                                     FIRSTSUNUM,LASTSUNUM) THEN
                                      BEGIN

                                                   EUSUJUNKALLOC(JNKPTR
                                          (TOPBLOCK),FIRSTEUNUM,
                                          LASTEUNUM,FIRSTSUNUM,
                                          LASTSUNUM);
                                        AMOUNTOFFREEDISK:=
                                         AMOUNTOFFREEDISK-NUMSEGS
                                         (TOPBLOCK);
                                        ALLOCATEONDISK:=TRUE;
                                        GO TO FIN;
                                      END;
                                    DEALLOCATELIST(PHASEDPTR
                            (TOPBLOCK));
                                    GO TO JUMP;
                                  END;
                         END;
JUMP:
                       DEALLOCATESU(VIRTUALSU(TOPBLOCK));
                     END;
                   DEALLOCATEEU(VIRTUALEU(TOPBLOCK));
                 END;
               DEALLOCATEEU(PHYSICALEU(TOPBLOCK));
      ALLOCATEONDISK:=FALSE;
FIN:
    END ALLOCATEONDISK;
%
END ALLOCATE;
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  E N D    O F    D I S K    F I L E    A L L O C A T O R .
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%
%
%
PROCEDURE READEM;
  BEGIN
  ALPHA   ARRAY ANN[0:29],ANP[0:21];
  ALPHA ARRAY ZEROUT[0:2];
    INTEGER CNT,A,B,C,D,E,F,G,LINK,LINK1,ZB,QT;
    FORMAT FF(2I10),FF1(5I10);
    LABEL RDD,LBLKS;
    ANN[27] := 1;
    LINK1 := 1;
RDD:READ(INN,FF,A,B);
    ANN[0].[19:5] := A;
```

```
    ANN[0].[24:24] := LINK := 8;
    FILL ZEROUT[*] WITH "00000000";
    IF ANN[27] NEQ 0 AND ANN[28] NEQ 0 THEN
        BEGIN
        FOR CNT := 1 STEP 1 UNTIL 3 DO
        REPLACE (ANN[CNT],0,ZEROUT[*],0,1);
        ZB := 13;
        QT := 1;
        GO TO LBLKS;
        END;
    ZB := QT := 0;
    READ(INN,3,ANP[*]);
          REPLACE(ANN[*],8,ANP[*],0,3);
    FOR CNT := 0 STEP 1 UNTIL 1 DO
        BEGIN
        READ(INN,FF,A,B);
        ANN[7+CNT].[1:23] := A;
        ANN[7+CNT].[24:24] := B;
        END;
LBLKS: FOR CNT := 0 STEP 1 UNTIL ZB DO
       BEGIN
        READ(INN,FF,A,B);
        ANN[QT+2*CNT].[1:23] := A;
        ANN[QT+2*CNT].[24:24] := B;
        READ(INN,FF1,C,D,E,F,G);
        ANN[QT+1+2*CNT].[7:8] := C;
        ANN[QT+1+2*CNT].[15:4] := D;
        ANN[QT+1+2*CNT].[19:8] := F;
        ANN[QT+1+2*CNT].[27:10] := F;
        ANN[QT+1+2*CNT].[37:11] := G;

        END;
      WRITE (PASSFILE[LINK1],30,ANN[*]);
    WRITE(SIMFILE,15,ANN[*]);
    IF LINK NEQ 0 THEN
        BEGIN
        LINK1 := LINK;
        GO TO RDD;
        END;
    READ (INN,10,THING[*]);
    END READFM;
%
%
%
PROCEDURE WRITEM;
    BEGIN
    FORMAT F("PTRBLK #",I4," IS ",                                    A4 ,A3,
        "PTR NO. ",I2," IT HAS ",I10," SEGS ",I10," FRAGS AND PTS TO PT
RBLK #",I4),
           F2("THE ABOVE LEVEL BLK PTS TO THE FOLLOWING"//) ,
           F3(X4,"PTRBLK #",I4," IS ",A4,A3,"PTR NO. ",I2,
           " IT HAS ",I10," SEGS ",I10," FRAGS AND PTS TO PTRBLK",I4),
        F4(X4,"THE ABOVE LEVEL BLK PTS TO THE FOLLOWING"//) ,
        F5(X8,"PTRBLK #",I4," IS ",                                   A4   ,A3
        ,"PTR NO. ",I2," IT HAS ",I10," SEGS ",I10," FRAGS AND PTS TO P
TR BLK #",I4),
        F6(X8,"THE ABOVE LEVEL BLK PTS TO THE FOLLOWING"//),
        F7(X12,"ASSNBLK #",I10,X5," PHASE =",I10," SEGS =",I10,X5,
   "NXT FRAG IN FILE IS AT ASSNBLK #",I10),
        F12(X50,"PHASED ASSNBLKS"//) ,
        F14(X50,"CONTIGUOUS ASSNBLKS"//) ,
```

```
        F8(X12,"ASSNBLK #",I10,X5,"SEGS=",I10,X5,
        "NXT FRAG IN FILE IS AT ASNBLK #",I10),
        F15(X50,"JUNK ASSNBLKS"//) ,
        F16(X50,"THE FILE REQUEST WAS"),
        F27("TOPBLK PTS TO"//),
        FRT("FILE BLOCK #",I10, " IS CALLED"),
        FRT1("NO. OF SEGS/RECD =",I10," TOTAL NO. OF SEGS =",I10,
          " THE PREPROCESS REQUEST PTR IS ",I10),
      FRT2(" THE POSTPROCESS PTR IS ",I10,
        " THE FIRST FRAG IN THE FILE IS AT  ASSNBLK #",I10//),
    FRT3("  IN TOPBLK, # OF SEGS =",I10,", NO. OF FRAGS =",I10)
      BOOLEAN DOO,RE,MI,DO1,RE1,DO2,DIDIT,DO22,RE11,MI1;
      ALPHA TYPE,WD;
     ALPHA ARRAY ALA[0:9];
     INTEGER ARRAY NXTEU[0:20],NXTSU[0:20],NXTTRK[0:20];
      INTEGER EUNUMBER,SUNUMBER,TRKNUMBER,VPP,SEGSS,FRAGSS,ANDY,
          ANDYEU,CN3,  ANDYSU,NM,     PANDA,PHSS,NXTTT,CNT,A,B,C,D,E,ZZ,
     CNT1,CNT2;
      LABEL ZZAZ,ZZAZ1,ZZAZ2,MAMA2,MAMA1,MAMA,LLVEUTR,LLVSUEU,LLVTRKSU,
          FINIT,HUBBA,CONFIN,HUBBA1,JNKFIN,HUBBA2,MORE1,BABA,BABA1,
          BABA2,MAMAA,MAMAA1,MAMAA2;
     DOO:=RE:=MI:=DO1:=RE1:=DO2:=DO22:=RE11:=MI1:=FALSE;
FOR CNT:=0 STEP 1 UNTIL 9 DO REPLACE(ALA[CNT],0,THING[CNT],0,1);
     FOR CNT := 0 STEP 1 UNTIL 2 DO
     WRITE(SIMFILE,*,PTRBLK[0,3+CNT]);
     WRITE(SIMFILE,F16);
     WRITE(SIMFILE,10,ALA[*]);
      FOR CNT:=1 STEP 1 UNTIL NXFIL DO
     BEGIN
     REPLACE(ALA[0],0,AFILNAME(CNT),0,1);
     REPLACE(ALA[1],0,BFILNAME(CNT),0,1);
     REPLACE(ALA[2],0,CFILNAME(CNT),0,1);
     A:= SEGSRECD(CNT);
     B:= TOTSEGS(CNT);
     C:= PREPTR(CNT);
     D:= POSTPTR(CNT); E:= FILASSN(CNT);
     WRITE(SIMFILE,FRT,CNT);
     WRITE(SIMFILE,3,ALA[*]);
     WRITE(SIMFILE,FRT1,A,B,C);  WRITE(SIMFILE,FRT2,D,E);
     END;
     NM := 1; A:= NUMSEGS(NM); B:= NUMFRAGS(NM);
     WRITE(SIMFILE,FRT3,A,B);
     CN3 := CNT1 := CNT2 := 0;
     ANDY := PHYPTR( NM);
     IF ANDY = 0 THEN GO TO LLVEUTR;
     WRITE(SIMFILE,F27);
ZZAZ:TYPE ← " EU";
      EUNUMBER ← NUMPTR(ANDY);
      VPP ← VPPTR(ANDY);
      SEGSS ← NUMSEGS(ANDY);
      FRAGSS ← NUMFRAGS(ANDY);
      CN3 := CN3 + 1;
      NXTEU[CN3]  ← NXTPTR(ANDY);
      ANDYEU ← ANDY;
      IF VPP=1 THEN WD:="PHYS" ELSE WD:="VIRT";
      WRITE(SIMFILE,F,ANDY,WD,TYPE,EUNUMBER,SEGSS,FRAGSS,NXTEU[CN3]);
      ANDY ← PHYPTR(ANDY);
      IF ANDY =0 THEN GO TO LLVSUEU;
     WRITE(SIMFILE,F2);
ZZAZ1:TYPE ← " SU";
      SUNUMBER ← NUMPTR(ANDY);
```

```
      VPP ← VPPTR(ANDY);
      SEGSS ← NUMSEGS(ANDY);
      FRAGSS ← NUMFRAGS(ANDY);
      CNT1 := CNT1 + 1;
      NXTSU[CNT1] ← NXTPTR(ANDY);
      ANDYSU ← ANDY;
      IF VPP=1 THEN WD:="PHYS" ELSE WD:="VIRT";
      WRITE(SIMFILE,F3,ANDY,WD,TYPE,SUNUMBER,SEGSS,FRAGSS,NXTSU[CNT1]);
      ANDY ← PHYPTR(ANDY);
     IF ANDY = 0 THEN GO TO LLVTRKSU;
     WRITE(SIMFILE,F4);
ZZAZ2:TYPE ← "TRK";
      TRKNUMBER ← NUMBPTR(ANDY);
      VPP ← VPPTR(ANDY);
      SEGSS ← NUMSEGS(ANDY);
      FRAGSS ← NUMFRAGS(ANDY);
      CNT2 := CNT2 + 1;
      NXTTRK[CNT2] ← NXTPTR(ANDY);
      IF VPP=1 THEN WD:="PHYS" ELSE WD:="VIRT";
     WRITE(SIMFILE,F5,ANDY,WD,TYPE,TRKNUMBER,SEGSS,FRAGSS,
          NXTTRK[CNT2]);
     ZZ := 3;
      GO TO FINIT;
LLVEUTB:
      ANDY := NXTPTR( NM);
      DOO := TRUE;
      IF ANDY ≠ 0 THEN BEGIN WRITE(SIMFILE,F27); GO TO ZZAZ; END;
MAMA2:
      ANDY := VSUPTR(NM);
      FE ← TRUE;
      IF ANDY ≠ 0 THEN BEGIN WRITE(SIMFILE,F27); GO TO ZZAZ1; END;
MAMA1:
      ANDY := VTRKPTR(NM);
      MI ← TRUE;
      IF ANDY ≠ 0 THEN BEGIN WRITE(SIMFILE,F27);GO TO ZZAZ2;END;
MAMAA2:ANDY := 1;
     MI1 := TRUE;
     ZZ := 0;
     GO TO FINIT;
LLVSUEU:
      ANDY ← VSUPTR(ANDYEU);
      DO1 ← TRUE;
      IF ANDY ≠ 0 THEN BEGIN WRITE(SIMFILE,F2); GO TO ZZAZ1; END;
MAMA:
      ANDY ← VTRKPTR(ANDYEU);
      FE1 ← TRUE;
      IF ANDY ≠ 0 THEN BEGIN WRITE(SIMFILE,F2); GO TO ZZAZ2; END;
MAMAA:ANDY := ANDYEU;
     FF11 := TRUE;
     ZZ := 1;
      ANDYEU := 0;
      GO TO FINIT;
LLVTRKSU:
      ANDY ← VTRKPTR(ANDYSU);
      DO2 ← TRUE;
      IF ANDY ≠ 0 THEN BEGIN WRITE(SIMFILE,F4); GO TO ZZAZ2; END;
MAMAA1:ZZ:=2;
     DO22:=TRUE;
      ANDY ← ANDYSU;
      ANDYSU := 0;
FINIT:DIDIT := TRUE;
```

```
      PANDA ← PHASPTR(ANDY);
     CNT := 0;
HURBA:
      IF PANDA =0 THEN GO TO CONFIN;
      PHSS ← PHNUM(PANDA);
      SEGSS ← SEGNUM(PANDA);
      NXTTT ← NXTFRAG(PANDA);
      IF DIDIT THEN BEGIN
     DIDIT := FALSE;
     CASE ZZ OF
          BEGIN
          WRITE(SIMFILE,F27);
          WRITE(SIMFILE,F2);
          WRITE(SIMFILE,F4);
          WRITE(SIMFILE,F6);
          END;
      END;
     IF CNT = 0 THEN
      WRITE(SIMFILE,F12);
      WRITE(SIMFILE,F7,PANDA,PHSS,SEGSS,NXTTT);
      PANDA ← NXTRL(PANDA);
     CNT := CNT + 1;
      GO TO HURBA;
CONFIN:
      PANDA ← CONTIGPTR(ANDY);
     CNT := 0;
HURBA1:
      IF PANDA = 0 THEN GO TO JNKFIN;
      SEGSS ← SEGNUM(PANDA);
      NXTTT ← NXTFRAG(PANDA);
     IF DIDIT THEN
          BEGIN
          DIDIT := FALSE;
     CASE ZZ OF
          BEGIN
          WRITE(SIMFILE,F27);
          WRITE(SIMFILE,F2);
          WRITE(SIMFILE,F4);
          WRITE(SIMFILE,F6);
          END;
          END;
     IF CNT = 0 THEN
      WRITE(SIMFILE,F14);
      WRITE(SIMFILE,F8,PANDA,SEGSS,NXTTT);
      PANDA ← NXTRL(PANDA);
     CNT := CNT + 1;
      GO TO HURBA1;
JNKFIN:
      PANDA ← JUNKPTR(ANDY);
      CNT := 0;
HURBA2:
      IF PANDA = 0 THEN GO TO MORE1;
      SEGSS ← SEGNUM(PANDA);
      NXTTT ← NXTFRAG(PANDA);
     IF DIDIT THEN
          BEGIN
          DIDIT := FALSE;
          CASE ZZ OF
          BEGIN
          WRITE(SIMFILE,F27);
          WRITE(SIMFILE,F2);
```

```
        WRITE(SIMFILE,F4);
        WRITE(SIMFILE,F6);
        END;
      END;
    IF CNT = 0 THEN
     WRITE(SIMFILE,F15);
     WRITE(SIMFILE,F8,PANDA,SEGSS,NXTTT);
     PANDA := NXTRL(PANDA);
     CNT :=  CNT + 1;
     GO TO HUBBA2;
MORE1: IF CNT2 NEQ 0 THEN
     BEGIN
     ANDY := NXTTRK[CNT2];
     CNT2 := CNT2 - 1;
    IF ANDY NEQ 0 THEN
     GO TO ZZAZ2 ELSE GO TO MORE1;
     END;
     IF DD22 THEN GO TO BABA;
     IF DD2 THEN IF ANDYSU NEQ 0 THEN GO TO MAMAA1 ELSE GO TO BABA;
     IF ANDYSU # 0 THEN GO TO LLVTRKSU;
BABA:
     IF RE11 THEN GO TO BABA1;
    IF DD2 THEN IF ANDYEU NEQ 0 THEN GO TO MAMAA ELSE GO TO BABA1;
     IF DD1 THEN IF ANDYEU NEQ 0 THEN GO TO MAMA ELSE GO TO  BABA1;
     IF ANDYEU # 0 THEN GO TO LLVSUEU;
BABA1: IF CNT1 NEQ 0 THEN
     BEGIN
     ANDY := NXTSU[CNT1];
     CNT1 := CNT1 - 1;
     IF ANDY NEQ 0 THEN BEGIN
     DD2 := DD22 := FALSE;
     GO TO ZZAZ1; END ELSE GO TO BABA1;
     END;
     IF MI1 THEN GO TO BABA2;
     IF MI THEN GO TO MAMAA2;
     IF RE THEN GO TO MAMA1;
     IF DDD THEN GO TO MAMA2;
     GO TO LIVEUTR;
BABA2: IF CN3 NEQ 0 THEN
     BEGIN
     ANDY := NXTEU[CN3];
     CN3 := CN3 - 1;
     IF ANDY NEQ 0 THEN BEGIN
     DD2 := DD1 := RE1 := RE11 := DD22 := FALSE;
     GO TO ZZAZ;
     END ELSE GO TO BABA2;
     END;
     END WRITEM;
$
```

D-3. SAMPLE OF EXECUTION OUTPUT

All of the output lines from the simulator itself are preceded by the word "SIM--". These lines contain the request number, queue information, time in a queue until allocation, simulated running time, processor time for procedures TREEBUILD and ALLOCATE, and the current system time.

```
00000005  1BFIL                    0000000000000000000000000801#H0000000000000009H0HYx0000009H001+0.00800000#0002x0.00804-
0001000#00000007O+K0000000000007O+K0000000000010+Yx80x0000000020?0x80x0000000030#00801H0000000408U0+0000000000609000138
00000000  1AFIL                   0+K0000000000010+Yx80x0000A01#H0000000000000009H0HYx0000009H001+0.00800000#0002x0.00804-


SIM--REQUEST     1 HAS JUST ENTERED QUEUE   8 AT SIMULATION TIME      0.
SIM--THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 0] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 1] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 2] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 3] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 4] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 5] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 6] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 7] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 8] IS     1.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 9] IS     0.

SIM--QUEUE[ 0]-     TIMESHARD IS    250 UNITS.  TIMESLICE IS    250 UNITS.
SIM--QUEUE[ 1]-     TIMESHARD IS    250 UNITS.  TIMESLICE IS    250 UNITS.
SIM--QUEUE[ 2]-     TIMESHARD IS    250 UNITS.  TIMESLICE IS    250 UNITS.
SIM--QUEUE[ 3]-     TIMESHARD IS    250 UNITS.  TIMESLICE IS    250 UNITS.
SIM--QUEUE[ 4]-     TIMESHARD IS    250 UNITS.  TIMESLICE IS    250 UNITS.
SIM--QUEUE[ 5]-     TIMESHARD IS    250 UNITS.  TIMESLICE IS    250 UNITS.
SIM--QUEUE[ 6]-     TIMESHARD IS    250 UNITS.  TIMESLICE IS    250 UNITS.
SIM--QUEUE[ 7]-     TIMESHARD IS    250 UNITS.  TIMESLICE IS    250 UNITS.
SIM--QUEUE[ 8]-     TIMESHARD IS    250 UNITS.  TIMESLICE IS    250 UNITS.
SIM--QUEUE[ 9]-     TIMESHARD IS    250 UNITS.  TIMESLICE IS    250 UNITS.


SIM--REQUEST     2 HAS JUST ENTERED QUEUE   8 AT SIMULATION TIME     28.
SIM--THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 0] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 1] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 2] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 3] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 4] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 5] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 6] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 7] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 8] IS     2.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 9] IS     0.
SIM--ALLOCATION ATTEMPT TO BE MADE ON QUEUE[ 8].
SIM--REQUEST     1 = PROCESSOR TIME FOR TREEBUILD IS    356.


000Y, 0H
0#02050>
0000600G
                                                    THE FILE REQUEST WAS
(PEU1(PSUO(PTRKO(P0:600 SEGMENTS))),SU1[P0:10,P10 MS:20),100C,500,200C)
FILE BLOCK #          1 IS CALLED
  1BFIL
NO. OF SEGS/RECD =          8 TOTAL NO. OF SEGS =      7960 THE PREPROCESS REQUEST PTR IS          0
 THE POSTPROCESS PTR IS          C THE FIRST FRAG IN THE FILE IS AT  ASSNBLK #          1


FILE BLOCK #          2 IS CALLED
```

```
  1AFIL
NO. OF SEGS/RECO =          8 TOTAL NO. OF SEGS =      7960 THE PREPROCESS REQUEST PTR IS          0
 THE POSTPROCESS PTR IS           0 THE FIRST FRAG IN THE FILE IS AT  ASSNBLK #         19


  IN TOPBLK, # OF SEGS =      14571, NO. OF FRAGS =         24
TOPBLK PTS TO

PTRBLK #   2 IS PHYS EUPTR NO.  1 IT HAS        1201 SEGS          3 FRAGS AND PTS TO PT RBLK #   6
THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

    PTRBLK #   3 IS PHYS SUPTR NO.  0 IT HAS        1200 SEGS          2 FRAGS AND PTS TO PTRBLK   8
    THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

        PTRBLK #   4 IS PHYSTRKPTR NO.  0 IT HAS        1200 SEGS,         2 FRAGS,AND PTS TO P TR BLK #   0
        THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                                PHASED ASSNBLKS

            ASSNBLK #         19      PHASE =          0 SEGS =        600     NXT FRAG IN FILE IS AT ASSNBLK #         20
            ASSNBLK #          1      PHASE =          0 SEGS =        600     NXT FRAG IN FILE IS AT ASSNBLK #          2
    PTRBLK #   8 IS PHYS SUPTR NO.  1 IT HAS          1 SEGS           1 FRAGS AND PTS TO PTRBLK   0
    THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

        PTRBLK #   9 IS PHYSTRKPTR NO.  1 IT HAS          1 SEGS,          1 FRAGS,AND PTS TO P TR BLK #   0
        THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                                CONTIGUOUS ASSNBLKS

            ASSNBLK #         13      SEGS=          1     NXT FRAG IN FILE IS AT ASNBLK #         14
TOPBLK PTS TO

PTRBLK #  10 IS VIRT EUPTR NO.  1 IT HAS          2 SEGS          1 FRAGS AND PTS TO PT RBLK #   0
THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

    PTRBLK #  11 IS VIRT SUPTR NO.  1 IT HAS          2 SEGS          1 FRAGS AND PTS TO PTRBLK   0
    THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

        PTRBLK #  12 IS VIRTTRKPTR NO.  1 IT HAS          2 SEGS,          1 FRAGS,AND PTS TO P TR BLK #   0
        THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                                JUNK ASSNBLKS

            ASSNBLK #         14      SEGS=          2     NXT FRAG IN FILE IS AT ASNBLK #         15
TOPBLK PTS TO

    PTRBLK #   5 IS VIRT SUPTR NO.  1 IT HAS        483 SEGS          5 FRAGS AND PTS TO PTRBLK  13
```

```
    THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                        PHASED ASSNBLKS

          ASSNBLK #        21      PHASE =       300 SEGS =       160     NXT FRAG IN FILE IS AT ASSNBLK #        22
          ASSNBLK #         3      PHASE =       300 SEGS =       160     NXT FRAG IN FILE IS AT ASSNBLK #         4
          ASSNBLK #        20      PHASE =         0 SEGS =        80     NXT FRAG IN FILE IS AT ASSNBLK #        21
          ASSNBLK #         2      PHASE =         0 SEGS =        80     NXT FRAG IN FILE IS AT ASSNBLK #         3
          ASSNBLK #        15      PHASE =       100 SEGS =         3     NXT FRAG IN FILE IS AT ASSNBLK #        16
PTRBLK #  13 IS VIRT SUPTR NO,  2 IT HAS          4 SEGS          1 FRAGS AND PTS TO PTRBLK   0
    THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                        CONTIGUOUS ASSNBLKS

          ASSNBLK #        16      SEGS=         4      NXT FRAG IN FILE IS AT ASNBLK #        17
TOPBLK PTS TO

       PTRBLK #  14 IS VIRTTRKPTR NO,  2 IT HAS          6 SEGS,         1 FRAGS,AND PTS TO P TR BLK #  15
       THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                        PHASED ASSNBLKS

          ASSNBLK #        17      PHASE =       200 SEGS =         6     NXT FRAG IN FILE IS AT ASSNBLK #        18
       PTRBLK #  15 IS VIRTTRKPTR NO,  1 IT HAS          8 SEGS,         1 FRAGS,AND PTS TO P TR BLK #   0
       THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                        JUNK ASSNBLKS

          ASSNBLK #        18      SEGS=         8      NXT FRAG IN FILE IS AT ASNBLK #         0
TOPBLK PTS TO

                                        CONTIGUOUS ASSNBLKS

          ASSNBLK #        24      SEGS=      1600      NXT FRAG IN FILE IS AT ASNBLK #         0
          ASSNBLK #         6      SEGS=      1600      NXT FRAG IN FILE IS AT ASNBLK #         7
          ASSNBLK #        22      SEGS=       800      NXT FRAG IN FILE IS AT ASNBLK #        23
          ASSNBLK #         4      SEGS=       800      NXT FRAG IN FILE IS AT ASNBLK #         5
                                        JUNK ASSNBLKS

          ASSNBLK #        23      SEGS=      4000      NXT FRAG IN FILE IS AT ASNBLK #        24
          ASSNBLK #         5      SEGS=      4000      NXT FRAG IN FILE IS AT ASNBLK #         6
PTRBLK #   6 IS PHYS EUPTR NO,  2 IT HAS         19 SEGS          2 FRAGS AND PTS TO PT RBLK #   7
THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                        PHASED ASSNBLKS

          ASSNBLK #         7      PHASE =       300 SEGS =         9     NXT FRAG IN FILE IS AT ASSNBLK #         8
```

```
                                                    JUNK ASSNBLKS

          ASSNBLK #          8     SEGS=         10     NXT FRAG IN FILE IS AT ASNBLK #          9
PTRBLK #   7 IS PHYS EUPTR NO.  4 IT HAS          48 SEGS          4 FRAGS AND PTS TO PT RBLK #    0
THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


                                                    CONTIGUOUS ASSNBLKS

          ASSNBLK #         12     SEGS=         12     NXT FRAG IN FILE IS AT ASNBLK #         13
          ASSNBLK #         11     SEGS=         12     NXT FRAG IN FILE IS AT ASNBLK #         12
          ASSNBLK #         10     SEGS=         12     NXT FRAG IN FILE IS AT ASNBLK #         11
          ASSNBLK #          9     SEGS=         12     NXT FRAG IN FILE IS AT ASNBLK #         10


SIM--REQUEST     1 WAS JUST ALLOCATED AFTER WAITING    28 UNITS OF SIMULATION TIME IN QUEUE    8
SIM--SIMTIME NOW IS     28.
SIM--REQUEST     1 -PROCESSOR TIME FOR ALLOCATION IS      228.
SIM--THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      1.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 0] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 1] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 2] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 3] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 4] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 5] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 6] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 7] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 8] IS     1.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 9] IS     0.


SIM--REQUEST     3 HAS JUST ENTERED QUEUE   9 AT SIMULATION TIME     67.
SIM--THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      1.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 0] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 1] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 2] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 3] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 4] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 5] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 6] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 7] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 8] IS     1.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 9] IS     1.
SIM--ALLOCATION ATTEMPT TO BE MADE ON QUEUE[ 8].
SIM--REQUEST     2 - PROCESSOR TIME FOR TREEBUILD IS    324.


000Y, 0H
0#02050>
00006000
                                                    THE FILE REQUEST WAS
(@EU1(@SU0(@TRK0(@0:600 SEGMENTS))),SU1[@0:10,@10 MS:20),100C,500,200C)
FILE BLOCK #          1 IS CALLED
  1BFIL
NO. OF SEGS/RECD =          8 TOTAL NO. OF SEGS =      7960 THE PREPROCESS REQUEST PTR IS          0
 THE POSTPROCESS PTR IS           0 THE FIRST FRAG IN THE FILE IS AT  ASSNBLK #          0
```

```
FILE BLOCK #          2 IS CALLED
  1AFIL
NO. OF SEGS/RECD =          8 TOTAL NO. OF SEGS =       7960 THE PREPROCESS REQUEST PTR IS          0
 THE POSTPROCESS PTR IS          0 THE FIRST FRAG IN THE FILE IS AT  ASSNBLK #         19


  IN TOPBLK, # OF SEGS =      14571, NO. OF FRAGS =         24
TOPBLK PTS TO


PTRBLK #   2 IS PHYS EUPTR NO.  1 IT HAS        1201 SEGS          3 FRAGS AND PTS TO PT RBLK #   6
THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

    PTRBLK #   3 IS PHYS SUPTR NO.  0 IT HAS        1200 SEGS          2 FRAGS AND PTS TO PTRBLK   8
    THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

        PTRBLK #   4 IS PHYSTRKPTR NO.  0 IT HAS        1200 SEGS,         2 FRAGS,AND PTS TO P TR BLK #   0
        THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                                PHASED ASSNBLKS

            ASSNBLK #         19      PHASE =          0 SEGS =        600     NXT FRAG IN FILE IS AT ASSNBLK #        20
            ASSNBLK #          1      PHASE =          0 SEGS =        600     NXT FRAG IN FILE IS AT ASSNBLK #         2
    PTRBLK #   8 IS PHYS SUPTR NO.  1 IT HAS          1 SEGS           1 FRAGS AND PTS TO PTRBLK   0
    THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

        PTRBLK #   9 IS PHYSTRKPTR NO.  1 IT HAS          1 SEGS,          1 FRAGS,AND PTS TO P TR BLK #   0
        THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                                CONTIGUOUS ASSNBLKS

            ASSNBLK #         13      SEGS=          1     NXT FRAG IN FILE IS AT ASNBLK #         14
TOPBLK PTS TO

PTRBLK #  10 IS VIRT EUPTR NO.  1 IT HAS          2 SEGS          1 FRAGS AND PTS TO PT RBLK #   0
THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

    PTRBLK #  11 IS VIRT SUPTR NO.  1 IT HAS          2 SEGS          1 FRAGS AND PTS TO PTRBLK   0
    THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

        PTRBLK #  12 IS VIRTTRKPTR NO.  1 IT HAS          2 SEGS,          1 FRAGS,AND PTS TO P TR BLK #   0
        THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                                JUNK ASSNBLKS

            ASSNBLK #         14      SEGS=          2     NXT FRAG IN FILE IS AT ASNBLK #         15
TOPBLK PTS TO
```

```
PTRBLK #   5 IS VIRT SUPTR NO.  1 IT HAS        483 SEGS          5 FRAGS AND PTS TO PTRBLK   13
THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                        PHASED ASSNBLKS

          ASSNBLK #        21      PHASE =       300 SEGS =        160      NXT FRAG IN FILE IS AT ASSNBLK #        22
          ASSNBLK #         3      PHASE =       300 SEGS =        160      NXT FRAG IN FILE IS AT ASSNBLK #         4
          ASSNBLK #        20      PHASE =         0 SEGS =         80      NXT FRAG IN FILE IS AT ASSNBLK #        21
          ASSNBLK #         2      PHASE =         0 SEGS =         80      NXT FRAG IN FILE IS AT ASSNBLK #         3
          ASSNBLK #        15      PHASE =       100 SEGS =          3      NXT FRAG IN FILE IS AT ASSNBLK #        16
PTRBLK #  13 IS VIRT SUPTR NO.  2 IT HAS          4 SEGS           1 FRAGS AND PTS TO PTRBLK    0
THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                        CONTIGUOUS ASSNBLKS

          ASSNBLK #        16      SEGS=         4      NXT FRAG IN FILE IS AT ASNBLK #        17
TOPBLK PTS TO

       PTRBLK #  14 IS VIRTTRKPTR NO.  2 IT HAS          6 SEGS,         1 FRAGS,AND PTS TO P TR BLK #  15
       THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                        PHASED ASSNBLKS

          ASSNBLK #        17      PHASE =       200 SEGS =          6      NXT FRAG IN FILE IS AT ASSNBLK #        18
       PTRBLK #  15 IS VIRTTRKPTR NO.  1 IT HAS          8 SEGS,         1 FRAGS,AND PTS TO P TR BLK #   0
       THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                        JUNK ASSNBLKS

          ASSNBLK #        18      SEGS=         8      NXT FRAG IN FILE IS AT ASNBLK #         6
TOPBLK PTS TO

                                        CONTIGUOUS ASSNBLKS

          ASSNBLK #        24      SEGS=      1600      NXT FRAG IN FILE IS AT ASNBLK #         1
          ASSNBLK #         6      SEGS=      1600      NXT FRAG IN FILE IS AT ASNBLK #         7
          ASSNBLK #        22      SEGS=       800      NXT FRAG IN FILE IS AT ASNBLK #        23
          ASSNBLK #         4      SEGS=       800      NXT FRAG IN FILE IS AT ASNBLK #         5
                                        JUNK ASSNBLKS

          ASSNBLK #        23      SEGS=      4000      NXT FRAG IN FILE IS AT ASNBLK #        24
          ASSNBLK #         5      SEGS=      4000      NXT FRAG IN FILE IS AT ASNBLK #         6
PTRBLK #   6 IS PHYS SUPTR NO.  2 IT HAS        19 SEGS          2 FRAGS AND PTS TO PT RBLK #   7
THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                        PHASED ASSNBLKS
```

```
            ASSNBLK #         7      PHASE =       300 SEGS =         9      NXT FRAG IN FILE IS AT ASSNBLK #         8
                                                   JUNK ASSNBLKS

            ASSNBLK #         8      SEGS=         10      NXT FRAG IN FILE IS AT ASSNBLK #         9
PTRBLK #   7 IS PHYS EUPTR NO.  4 IT HAS          48 SEGS          4 FRAGS AND PTS TO PT RBLK #   0
THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


                                                   CONTIGUOUS ASSNBLKS

            ASSNBLK #        12      SEGS=         12      NXT FRAG IN FILE IS AT ASSNBLK #        13
            ASSNBLK #        11      SEGS=         12      NXT FRAG IN FILE IS AT ASSNBLK #        12
            ASSNBLK #        10      SEGS=         12      NXT FRAG IN FILE IS AT ASSNBLK #        11
            ASSNBLK #         9      SEGS=         12      NXT FRAG IN FILE IS AT ASSNBLK #        10


SIM--REQUEST     2 WAS JUST ALLOCATED AFTER WAITING    39 UNITS OF SIMULATION TIME IN QUEUE   8
SIM--SIMTIME NOW IS      67.
SIM--REQUEST     2 -PROCESSOR TIME FOR ALLOCATION IS       225.
SIM--THE NUMBER OF REQUESTS ALREADY ALLOCATED IS       2.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 0) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 1) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 2) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 3) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 4) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 5) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 6) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 7) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 8) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 9) IS     1.


SIM--REQUEST     4 WAS JUST ENTERED QUEUE   1 AT SIMULATION TIME     145.
SIM--THE NUMBER OF REQUESTS ALREADY ALLOCATED IS       2.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 0) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 1) IS     1.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 2) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 3) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 4) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 5) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 6) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 7) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 8) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 9) IS     1.
SIM--ALLOCATION ATTEMPT TO BE MADE ON QUEUE( 1).
SIM--REQUEST     4 - PROCESSOR TIME FOR TREEBUILD IS     329.


000Y, 0H
0#02050>
00006006
                                                   THE FILE REQUEST WAS
(PEU1(PSU0(6TRK0(P0:600 SEGMENTS))),SU1(P0:10,P10 MS:20),100C,500,200C)
FILE BLOCK #         1 IS CALLED
```

```
  1BFIL
NO. OF SEGS/RECD =          8 TOTAL NO. OF SEGS =      7960 THE PREPROCESS REQUEST PTR IS          0
 THE POSTPROCESS PTR IS          0 THE FIRST FRAG IN THE FILE IS AT  ASSNBLK #          0


FILE BLOCK #          2 IS CALLED
  1AFIL
NO. OF SEGS/RECD =          8 TOTAL NO. OF SEGS =      7960 THE PREPROCESS REQUEST PTR IS          0
 THE POSTPROCESS PTR IS          0 THE FIRST FRAG IN THE FILE IS AT  ASSNBLK #         19


  IN TOPBLK, # OF SEGS =      14571, NO. OF FRAGS =         24
TOPBLK PTS TO


PTRBLK #   2 IS PHYS EUPTR NO.  1 IT HAS        1201 SEGS          3 FRAGS AND PTS TO PT RBLK #   6
THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


    PTRBLK #   3 IS PHYS SUPTR NO.  0 IT HAS        1200 SEGS          2 FRAGS AND PTS TO PTRBLK   8
    THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


        PTRBLK #   4 IS PHYSTRKPTR NO.  0 IT HAS        1200 SEGS,         2 FRAGS,AND PTS TO P TR BLK #   0
        THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


                                                PHASED ASSNBLKS


            ASSNBLK #         19      PHASE =          0 SEGS =       600     NXT FRAG IN FILE IS AT ASSNBLK #         20
            ASSNBLK #          1      PHASE =          0 SEGS =       600     NXT FRAG IN FILE IS AT ASSNBLK #          2
    PTRBLK #   8 IS PHYS SUPTR NO.  1 IT HAS          1 SEGS           1 FRAGS AND PTS TO PTRBLK   0
    THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


        PTRBLK #   9 IS PHYSTRKPTR NO.  1 IT HAS          1 SEGS,          1 FRAGS,AND PTS TO P TR BLK #   0
        THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


                                                CONTIGUOUS ASSNBLKS


            ASSNBLK #         13      SEGS=          1     NXT FRAG IN FILE IS AT ASNBLK #         14
TOPBLK PTS TO


PTRBLK #  10 IS VIRT EUPTR NO.  1 IT HAS          2 SEGS          1 FRAGS AND PTS TO PT RBLK #   0
THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


    PTRBLK #  11 IS VIRT SUPTR NO.  1 IT HAS          2 SEGS          1 FRAGS AND PTS TO PTRBLK   0
    THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


        PTRBLK #  12 IS VIRTTRKPTR NO.  1 IT HAS          2 SEGS,          1 FRAGS,AND PTS TO P TR BLK #   0
        THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


                                                JUNK ASSNBLKS
```

```
          ASSNBLK #        14      SEGS=          2      NXT FRAG IN FILE IS AT ASNBLK #        15
TOPBLK PTS TO

    PTRBLK #   5 IS VIRT SUPTR NO.  1 IT HAS          483 SEGS          5 FRAGS AND PTS TO PTRBLK  13
    THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                                PHASED ASSNBLKS

          ASSNBLK #        21      PHASE =      300 SEGS =       160      NXT FRAG IN FILE IS AT ASSNBLK #        22
          ASSNBLK #         3      PHASE =      300 SEGS =       160      NXT FRAG IN FILE IS AT ASSNBLK #         4
          ASSNBLK #        20      PHASE =        0 SEGS =        80      NXT FRAG IN FILE IS AT ASSNBLK #        21
          ASSNBLK #         2      PHASE =        0 SEGS =        80      NXT FRAG IN FILE IS AT ASSNBLK #         3
          ASSNBLK #        15      PHASE =      100 SEGS =         3      NXT FRAG IN FILE IS AT ASSNBLK #        16
    PTRBLK #  13 IS VIRT SUPTR NO.  2 IT HAS            4 SEGS          1 FRAGS AND PTS TO PTRBLK   0
    THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                                CONTIGUOUS ASSNBLKS

          ASSNBLK #        16      SEGS=          4      NXT FRAG IN FILE IS AT ASNBLK #        17
TOPBLK PTS TO

       PTRBLK #  14 IS VIRTTRKPTR NO.  2 IT HAS          6 SEGS,          1 FRAGS,AND PTS TO P TR BLK #  15
       THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                                PHASED ASSNBLKS

          ASSNBLK #        17      PHASE =      200 SEGS =         6      NXT FRAG IN FILE IS AT ASSNBLK #        18
       PTRBLK #  15 IS VIRTTRKPTR NO.  1 IT HAS          8 SEGS,          1 FRAGS,AND PTS TO P TR BLK #   0
       THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                                JUNK ASSNBLKS

          ASSNBLK #        18      SEGS=          8      NXT FRAG IN FILE IS AT ASNBLK #         0
TOPBLK PTS TO

                                                CONTIGUOUS ASSNBLKS

          ASSNBLK #        24      SEGS=       1600      NXT FRAG IN FILE IS AT ASNBLK #         1
          ASSNBLK #         6      SEGS=       1600      NXT FRAG IN FILE IS AT ASNBLK #         7
          ASSNBLK #        22      SEGS=        800      NXT FRAG IN FILE IS AT ASNBLK #        23
          ASSNBLK #         4      SEGS=        800      NXT FRAG IN FILE IS AT ASNBLK #         5
                                                JUNK ASSNBLKS

          ASSNBLK #        23      SEGS=       4000      NXT FRAG IN FILE IS AT ASNBLK #        24
          ASSNBLK #         5      SEGS=       4000      NXT FRAG IN FILE IS AT ASNBLK #         6
PTRBLK #   6 IS PHYS SUPTR NO.  2 IT HAS           19 SEGS          2 FRAGS AND PTS TO PT RBLK #   7
THE ABOVE LEVEL BLK PTS TO THE FOLLOWING
```

```
                                        PHASED ASSNBLKS

          ASSNBLK #        7      PHASE =      300 SEGS =        9     NXT FRAG IN FILE IS AT ASSNBLK #        8
                                                JUNK ASSNBLKS

          ASSNBLK #        8      SEGS=        10     NXT FRAG IN FILE IS AT ASNBLK #        9
PTRBLK #   7 IS PHYS EUPTR NO.  4 IT HAS         48 SEGS          4 ERAGS AND PTS TO PT RBLK #   0
THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


                                        CONTIGUOUS ASSNBLKS

          ASSNBLK #       12      SEGS=        12     NXT FRAG IN FILE IS AT ASNBLK #       13
          ASSNBLK #       11      SEGS=        12     NXT FRAG IN FILE IS AT ASNBLK #       12
          ASSNBLK #       10      SEGS=        12     NXT FRAG IN FILE IS AT ASNBLK #       11
          ASSNBLK #        9      SEGS=        12     NXT FRAG IN FILE IS AT ASNBLK #       10


SIM--REQUEST     4 WAS JUST ALLOCATED AFTER WAITING      0 UNITS OF SIMULATION TIME IN QUEUE   1
SIM--SIMTIME NOW IS    145.
SIM--REQUEST     4 -PROCESSOR TIME FOR ALLOCATION IS       155.
SIM--THE NUMBER OF REQUESTS ALREADY ALLOCATED IS      3.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 0) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 1) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 2) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 3) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 4) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 5) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 6) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 7) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 8) IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE( 9) IS     1.


SIM--REQUEST   1 IS FINISHED AFTER   123 UNITS OF TIME.

SIM--QUEUE( 0)-      TIMESHARD IS     250 UNITS.  TIMESLICE IS     250 UNITS.
SIM--QUEUE( 1)-      TIMESHARD IS     250 UNITS.  TIMESLICE IS     250 UNITS.
SIM--QUEUE( 2)-      TIMESHARD IS     250 UNITS.  TIMESLICE IS     250 UNITS.
SIM--QUEUE( 3)-      TIMESHARD IS     250 UNITS.  TIMESLICE IS     250 UNITS.
SIM--QUEUE( 4)-      TIMESHARD IS     250 UNITS.  TIMESLICE IS     250 UNITS.
SIM--QUEUE( 5)-      TIMESHARD IS     250 UNITS.  TIMESLICE IS     250 UNITS.
SIM--QUEUE( 6)-      TIMESHARD IS     250 UNITS.  TIMESLICE IS     250 UNITS.
SIM--QUEUE( 7)-      TIMESHARD IS     250 UNITS.  TIMESLICE IS     250 UNITS.
SIM--QUEUE( 8)-      TIMESHARD IS     250 UNITS.  TIMESLICE IS     127 UNITS.
SIM--QUEUE( 9)-      TIMESHARD IS     250 UNITS.  TIMESLICE IS     250 UNITS.
SIM--ALLOCATION ATTEMPT TO BE MADE ON QUEUE( 9).
SIM--REQUEST     3 - PROCESSOR TIME FOR TREEBUILD IS     322.


000Y, 0H
0#02050>
00006006
```

```
                                                    THE FILE REQUEST WAS
(PEU1(PSUO(PTRKO(P0:600 SEGMENTS))),SUI(P0:10,P10 MS:20),100C,500,200C)
FILE BLOCK #          1 IS CALLED
  IBFIL
NO. OF SEGS/RECD =          8 TOTAL NO. OF SEGS =       7960 THE PREPROCESS REQUEST PTR IS          0
 THE POSTPROCESS PTR IS           0 THE FIRST FRAG IN THE FILE IS AT  ASSNBLK #          0


FILE BLOCK #          2 IS CALLED
  IAFIL
NO. OF SEGS/RECD =          8 TOTAL NO. OF SEGS =       7960 THE PREPROCESS REQUEST PTR IS          0
 THE POSTPROCESS PTR IS           0 THE FIRST FRAG IN THE FILE IS AT  ASSNBLK #         19


  IN TOPBLK, # OF SEGS =      14571, NO. OF FRAGS =         24
TOPBLK PTS TO


PTRBLK #   2 IS PHYS EUPTR NO.  1 IT HAS        1201 SEGS          3 FRAGS AND PTS TO PT RBLK #   6
THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


    PTRBLK #   3 IS PHYS SUPTR NO.  0 IT HAS        1200 SEGS          2 FRAGS AND PTS TO PTRBLK   8
    THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


        PTRBLK #   4 IS PHYSTRKPTR NO.  0 IT HAS        1200 SEGS,         2 FRAGS,AND PTS TO P TR BLK #   0
        THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


                                                PHASED ASSNBLKS


            ASSNBLK #         19      PHASE =          0 SEGS =        600     NXT FRAG IN FILE IS AT ASSNBLK #         20
            ASSNBLK #          1      PHASE =          0 SEGS =        600     NXT FRAG IN FILE IS AT ASSNBLK #          2
    PTRBLK #   8 IS PHYS SUPTR NO.  1 IT HAS          1 SEGS          1 FRAGS AND PTS TO PTRBLK   0
    THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


        PTRBLK #   9 IS PHYSTRKPTR NO.  1 IT HAS          1 SEGS,         1 FRAGS,AND PTS TO P TR BLK #   0
        THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


                                                CONTIGUOUS ASSNBLKS


            ASSNBLK #         13      SEGS=          1     NXT FRAG IN FILE IS AT ASNBLK #         14
TOPBLK PTS TO


PTRBLK #  10 IS VIRT EUPTR NO.  1 IT HAS          2 SEGS          1 FRAGS AND PTS TO PT RBLK #   0
THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


    PTRBLK #  11 IS VIRT SUPTR NO.  1 IT HAS          2 SEGS          1 FRAGS AND PTS TO PTRBLK   0
    THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


        PTRBLK #  12 IS VIRTTRKPTR NO.  1 IT HAS          2 SEGS,         1 FRAGS,AND PTS TO P TR BLK #   0
        THE ABOVE LEVEL BLK PTS TO THE FOLLOWING
```

```
                                        JUNK ASSNBLKS

          ASSNBLK #        14      SEGS=         2     NXT FRAG IN FILE IS AT ASNBLK #        15
TOPBLK PTS TO

    PTRBLK #   5 IS VIRT SUPTR NO.  1 IT HAS        483 SEGS          5 FRAGS AND PTS TO PTRBLK  13
    THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                        PHASED ASSNBLKS

          ASSNBLK #        21      PHASE =       300 SEGS =       160     NXT FRAG IN FILE IS AT ASSNBLK #       22
          ASSNBLK #         3      PHASE =       300 SEGS =       160     NXT FRAG IN FILE IS AT ASSNBLK #        4
          ASSNBLK #        20      PHASE =         0 SEGS =        80     NXT FRAG IN FILE IS AT ASSNBLK #       21
          ASSNBLK #         2      PHASE =         0 SEGS =        80     NXT FRAG IN FILE IS AT ASSNBLK #        3
          ASSNBLK #        15      PHASE =       100 SEGS =         3     NXT FRAG IN FILE IS AT ASSNBLK #       16
    PTRBLK #  13 IS VIRT SUPTR NO.  2 IT HAS          4 SEGS          1 FRAGS AND PTS TO PTRBLK   0
    THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                        CONTIGUOUS ASSNBLKS

          ASSNBLK #        16      SEGS=         4     NXT FRAG IN FILE IS AT ASNBLK #        17
TOPBLK PTS TO

       PTRBLK #  14 IS VIRTTRKPTR NO.  2 IT HAS          6 SEGS,         1 FRAGS,AND PTS TO P TR BLK #  15
       THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                        PHASED ASSNBLKS

          ASSNBLK #        17      PHASE =       200 SEGS =         6     NXT FRAG IN FILE IS AT ASSNBLK #       18
       PTRBLK #  15 IS VIRTTRKPTR NO.  1 IT HAS          8 SEGS,         1 FRAGS,AND PTS TO P TR BLK #   0
       THE ABOVE LEVEL BLK PTS TO THE FOLLOWING

                                        JUNK ASSNBLKS

          ASSNBLK #        18      SEGS=         8     NXT FRAG IN FILE IS AT ASNBLK #         0
TOPBLK PTS TO

                                        CONTIGUOUS ASSNBLKS

          ASSNBLK #        24      SEGS=      1600     NXT FRAG IN FILE IS AT ASNBLK #         1
          ASSNBLK #         6      SEGS=      1600     NXT FRAG IN FILE IS AT ASNBLK #         7
          ASSNBLK #        22      SEGS=       800     NXT FRAG IN FILE IS AT ASNBLK #        23
          ASSNBLK #         4      SEGS=       800     NXT FRAG IN FILE IS AT ASNBLK #         5
                                        JUNK ASSNBLKS

          ASSNBLK #        23      SEGS=      4000     NXT FRAG IN FILE IS AT ASNBLK #        24
```

```
          ASSNBLK #         5      SEGS=       4000      NXT FRAG IN FILE IS AT ASNBLK #          6
PTRBLK #   6 IS PHYS EUPTR NO.  2 IT HAS          19 SEGS           2 FRAGS AND PTS TO PT RBLK #    7
THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


                                                PHASED ASSNBLKS

          ASSNBLK #         7      PHASE =        300 SEGS =          9      NXT FRAG IN FILE IS AT ASSNBLK #          8
                                                JUNK ASSNBLKS


          ASSNBLK #         8      SEGS=         10      NXT FRAG IN FILE IS AT ASNBLK #          9
PTRBLK #   7 IS PHYS EUPTR NO.  4 IT HAS          48 SEGS           4 FRAGS AND PTS TO PT RBLK #    0
THE ABOVE LEVEL BLK PTS TO THE FOLLOWING


                                                CONTIGUOUS ASSNBLKS

          ASSNBLK #        12      SEGS=         12      NXT FRAG IN FILE IS AT ASNBLK #         13
          ASSNBLK #        11      SEGS=         12      NXT FRAG IN FILE IS AT ASNBLK #         12
          ASSNBLK #        10      SEGS=         12      NXT FRAG IN FILE IS AT ASNBLK #         11
          ASSNBLK #         9      SEGS=         12      NXT FRAG IN FILE IS AT ASNBLK #         10


SIM--REQUEST     3 WAS JUST ALLOCATED AFTER WAITING     84 UNITS OF SIMULATION TIME IN QUEUE    9
SIM--SIMTIME NOW IS     151.
SIM--REQUEST     3 -PROCESSOR TIME FOR ALLOCATION IS        85.
SIM--THE NUMBER OF REQUESTS ALREADY ALLOCATED IS       3.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 0] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 1] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 2] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 3] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 4] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 5] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 6] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 7] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 8] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 9] IS     0.


SIM--REQUEST     5 HAS JUST ENTERED QUEUE    1 AT SIMULATION TIME     174.
SIM--THE NUMBER OF REQUESTS ALREADY ALLOCATED IS       3.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 0] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 1] IS     1.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 2] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 3] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 4] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 5] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 6] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 7] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 8] IS     0.
SIM--THE NUMBER OF REQUESTS IN QUEUE[ 9] IS     0.
SIM--ALLOCATION ATTEMPT TO BE MADE ON QUEUE[ 1].


########  -OPRTR OS-ED SIMULA /IADISK = 1, S = 110, A =  10  ##########################################################################
```

LIST OF REFERENCES

[1] Dahl, O.J. and Nygaard, K., "SIMULA--An ALGOL-Based Simulation Language," Comm. ACM, 9. (Sept., 1966) pp. 671-678

[2] ILLIAC IV Tape Number 34, SIMULA/MANUAL (provides a listing of the B5500 Implementation of SIMULA, 1966.)

[3] Knuth, Donald E., "Dynamic Storage Allocation," Fundamental Algorithms, Vol. 1, Addison-Wesley Publishing Company, (1968) pp. 435-455.

[4] Alsberg, Peter A. and Mills, Carlton R., "The Structure of the ILLIAC IV Operating System," Proc. Second Symposium on Operating Systems Principles, Princeton, Oct. 1969.

[5] Parnas, David L. and Darringer, John A., "SODAS and a Methodology for System Design," Proc. Fall Joint Computer Conference (1967) pp. 449-474.

[6] Parnas, David L. "More on Simulation Languages and Design Methodology for Computer Systems," Proc. Spring Joint Computer Conference (1969) pp. 739-743.

[7] Alsberg, Peter A., et al., "A Description of the ILLIAC IV Operating System," ILLIAC IV Document No. 212, File No. 791, Department of Computer Science, University of Illinois at Urbana-Champaign (November 1, 1968).

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Department of Computer Science University of Illinois at Urbana-Champaign Urbana, Illinois 61801	UNCLASSIFIED
	2b. GROUP

REPORT TITLE

A SIMULATION STUDY OF A DISK STORAGE ALLOCATION SYSTEM

DESCRIPTIVE NOTES (Type of report and inclusive dates)

Research Report

AUTHOR(S) (First name, middle initial, last name)

Judy Ann Lender

REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
April 30, 1970	116	7

CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
USAF 30(602)-4144	DCS Report No. 380
PROJECT NO. 46-26-15-305	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) ILLIAC IV Document No. 210

DISTRIBUTION STATEMENT

Qualified requesters may obtain copies from DCS.

SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
NONE	Rome Air Development Center Griffiss Air Force Base Rome, New York 13440

ABSTRACT

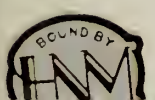
This report gives a discussion of the field of simulation. The use of the simulation language SIMULA is then used to program general disk storage allocation simulators with application for use in testing allocation algorithms for the ILLIAC IV disk file allocator. Finally the concept of system design by using simulation in the design phase at various design levels is presented, with emphasis on using SIMULA for design from the hardware level upward.

D FORM 1 NOV 65 1473

UNCLASSIFIED

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
ILLIAC IV						
SIMULA						
Simulation						
Storage allocation						
Disk file allocation						
Systems design						


BOUND BY
HW